from pathlib import Path

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

def categorizar\_rve(rve):

    if pd.isna(rve):

        return "Sin baseline suficiente"

    elif rve > 250:

        return ">250% (posible baseline bajo o pico)"

    elif rve < 50:

        return "<50% (muy por debajo)"

    else:

        return "Normal"

# Configuración general

plt.style.use('ggplot')

sns.set(rc={'figure.figsize':(10,6)})

# Ruta a la base de datos

DB\_PATH = Path(r'C:\\Users\\Nico\\Desktop\\DATA SCIENCE\\PP- VOLUNTAREADO\\chivas-ml\\data\\external\\chivas\_dw.sqlite')

conn = sqlite3.connect(str(DB\_PATH))

df\_ent = pd.read\_sql('SELECT \* FROM DB\_Entrenamientos', conn)

df\_part = pd.read\_sql\_query("""

    SELECT

        "id\_jugador",

        "Rival",

        "Fecha",

        "Local\_Visitante",

        "Duracion\_min",

        "Distancia\_total",

        "HSR\_abs\_m",

        "HSR\_rel\_m",

        "HMLD\_m",

        "Sprints\_distancia\_m",

        "Sprints\_cantidad",

        "Acc\_3",

        "Dec\_3",

        "Carga\_Explosiva",

        "Carga\_Sostenida",

        "Carga\_Regenerativa",

        "Rendimiento\_Partido",

        "Rendimiento\_vs\_Entreno"

    FROM DB\_Partidos

    """, conn)

# Cargar tabla jugadores

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

# Normalizar nombres de columnas

df\_jug.columns = df\_jug.columns.str.strip().str.lower().str.replace(' ', '\_')

# Unir entrenamientos con nombres de jugadores

df\_part = df\_part.merge(df\_jug[['id\_jugador','nombre','linea']], on='id\_jugador', how='left')

#Agregamos la columna RvE\_flag

df\_part["RvE\_flag"] = df\_part["Rendimiento\_vs\_Entreno"].apply(categorizar\_rve)

# Definí el orden que querés

column\_order = [

    "Rival", "Fecha", "nombre", "linea", "Local\_Visitante", "Duracion\_min",

    "Distancia\_total", "HSR\_abs\_m", "HSR\_rel\_m", "HMLD\_m",

    "Sprints\_distancia\_m", "Sprints\_cantidad",

    "Acc\_3", "Dec\_3",

    "Carga\_Explosiva", "Carga\_Sostenida", "Carga\_Regenerativa",

    "Rendimiento\_Partido", "Rendimiento\_vs\_Entreno", "RvE\_flag"

]

# Reordenar (y mantener las que no estén en column\_order al final)

df\_part = df\_part[[col for col in column\_order if col in df\_part.columns] +

                  [c for c in df\_part.columns if c not in column\_order]]

df\_part.dtypes

Rival object

Fecha object

nombre object

linea object

Local\_Visitante object

Duracion\_min float64

Distancia\_total int64

HSR\_abs\_m float64

HSR\_rel\_m float64

HMLD\_m int64

Sprints\_distancia\_m int64

Sprints\_cantidad int64

Acc\_3 int64

Dec\_3 int64

Carga\_Explosiva float64

Carga\_Sostenida float64

Carga\_Regenerativa float64

Rendimiento\_Partido float64

Rendimiento\_vs\_Entreno float64

RvE\_flag object

id\_jugador int64

dtype: object

# Aseguramos que Fecha sea datetime

df\_part["Fecha"] = pd.to\_datetime(df\_part["Fecha"], errors="coerce")

# Ordenamos por fecha

df\_sorted = df\_part.sort\_values("Fecha")

# Creamos una columna "Rival\_ordenado" que respete el orden cronológico

df\_sorted["Rival\_ordenado"] = pd.Categorical(

    df\_sorted["Rival"],

    categories=df\_sorted.sort\_values("Fecha")["Rival"].unique(),

    ordered=True

)

plt.figure(figsize=(12,6))

sns.lineplot(

    data=df\_sorted,

    x="Rival\_ordenado",

    y="Rendimiento\_Partido",

    hue="linea",

    marker="o"

)

plt.title("Evolución temporal del rendimiento por línea (orden real de partidos)")

plt.ylabel("Rendimiento (0-100)")

plt.xlabel("Rival (orden cronológico)")

plt.legend(title="Línea")

plt.show()

Gráfico, Gráfico radial

El contenido generado por IA puede ser incorrecto.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.dates as mdates

# --- Normalizar nombres de columnas por si vienen en snake\_case/minúsculas ---

cols = {c.lower(): c for c in df\_part.columns}

def col(name):  # devuelve el nombre real de la columna en el DF

    return cols.get(name.lower(), name)

# Asegurar tipos y ordenar

df\_part[col("Fecha")] = pd.to\_datetime(df\_part[col("Fecha")], errors="coerce")

dfp = df\_part.dropna(subset=[col("Fecha")]).copy()

dfp = dfp.sort\_values(col("Fecha"))

# --- Agregar por Partido (Fecha) x Línea: promedio de CE, CS, CR ---

agg = (dfp

       .groupby([col("Fecha"), col("Rival"), col("linea")], as\_index=False)

       .agg(CE=(col("Carga\_Explosiva"), "mean") if col("Carga\_Explosiva") in dfp.columns else ("Carga\_Explosiva", "mean"),

            CS=(col("Carga\_Sostenida"), "mean") if col("Carga\_Sostenida") in dfp.columns else ("Carga\_Sostenida", "mean"),

            CR=(col("Carga\_Regenerativa"), "mean") if col("Carga\_Regenerativa") in dfp.columns else ("Carga\_Regenerativa", "mean")))

# Orden temporal y etiquetas Fecha + Rival

agg = agg.sort\_values(col("Fecha"))

fecha\_col = col("Fecha")

rival\_col = col("Rival")

linea\_col = col("linea")

# build etiquetas "YYYY-MM-DD\nRival" por fecha (modo del rival por si hay duplicados)

rival\_por\_fecha = (agg[[fecha\_col, rival\_col]]

                   .drop\_duplicates(subset=[fecha\_col])

                   .set\_index(fecha\_col)[rival\_col])

fechas\_orden = agg[fecha\_col].drop\_duplicates().tolist()

tick\_labels = [f"{f.strftime('%Y-%m-%d')}\n{rival\_por\_fecha.get(f, '')}" for f in fechas\_orden]

# Helper: gráfico de barras agrupadas para una métrica

def plot\_barras\_agrupadas(df, value\_col, title, ylabel):

    # pivot: filas=Fecha, cols=Línea, valores=value\_col

    pvt = (df.pivot\_table(index=fecha\_col, columns=linea\_col, values=value\_col, aggfunc='mean')

             .reindex(index=fechas\_orden))

    lineas = list(pvt.columns)

    x = np.arange(len(pvt.index), dtype=float)

    width = 0.8 / max(1, len(lineas))  # ancho de cada barra

    fig, ax = plt.subplots(figsize=(12, 6))

    for i, lin in enumerate(lineas):

        y = pvt[lin].values

        ax.bar(x + i\*width - 0.4 + width/2, y, width=width, label=str(lin))

    # ejes/etiquetas

    ax.set\_title(title)

    ax.set\_ylabel(ylabel)

    ax.set\_xlabel("Fecha (debajo: Rival)")

    ax.set\_xticks(x)

    ax.set\_xticklabels(tick\_labels, rotation=0, ha='center')

    ax.legend(title="Línea", ncol=min(4, len(lineas)))

    ax.grid(axis='y', alpha=0.3)

    plt.tight\_layout()

    plt.show()

# --- Graficar CE, CS, CR (tres gráficos, uno por métrica) ---

plot\_barras\_agrupadas(agg, "CE", "Carga Explosiva por línea y partido", "Carga\_Explosiva (promedio por línea)")

plot\_barras\_agrupadas(agg, "CS", "Carga Sostenida por línea y partido", "Carga\_Sosotenida (promedio por línea)")

plot\_barras\_agrupadas(agg, "CR", "Carga Regenerativa por línea y partido", "Carga\_Regenerativa (promedio por línea)")

Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.

# --- asegurar datetime y ordenar ---

df\_part["Fecha"] = pd.to\_datetime(df\_part["Fecha"], errors="coerce")

df\_part = df\_part.sort\_values(["id\_jugador", "Fecha"])

# --- días de descanso por jugador ---

df\_part["days\_since\_last"] = df\_part.groupby("id\_jugador")["Fecha"].diff().dt.days

# --- resumen por partido (por fecha) ---

def \_mode\_or\_first(s):

    s = s.dropna()

    if s.empty:

        return np.nan

    # modo si hay, si no, primero

    try:

        m = s.mode()

        return m.iloc[0] if not m.empty else s.iloc[0]

    except Exception:

        return s.iloc[0]

df\_match\_summary = (df\_part

    .groupby("Fecha")

    .agg(

        Rival=("Rival", \_mode\_or\_first),

        Rend\_Partido\_Prom=("Rendimiento\_Partido", "mean"),

        RvE\_prom=("Rendimiento\_vs\_Entreno", "mean"),

        descanso\_prom=("days\_since\_last", "mean"),

        descanso\_min=("days\_since\_last", "min")

    )

    .reset\_index()

    .sort\_values("Fecha")

)

# --- etiquetas Fecha + Rival para los ticks ---

tick\_labels = [f"{d.strftime('%Y-%m-%d')}\n{r}" for d, r in zip(df\_match\_summary["Fecha"], df\_match\_summary["Rival"])]

# --- gráfico combinado: líneas + barras ---

fig, ax1 = plt.subplots(figsize=(12,6))

# líneas (eje izquierdo)

sns.lineplot(data=df\_match\_summary, x="Fecha", y="Rend\_Partido\_Prom", marker="o", label="Rendimiento promedio", ax=ax1)

sns.lineplot(data=df\_match\_summary, x="Fecha", y="RvE\_prom", marker="s", label="RvE promedio", ax=ax1)

ax1.set\_ylabel("Rendimiento / RvE")

ax1.set\_xlabel("Fecha (debajo: Rival)")

ax1.set\_title("Rendimiento en partidos vs días de descanso")

# formato de fechas en x

ax1.xaxis.set\_major\_formatter(mdates.DateFormatter('%Y-%m-%d'))

ax1.set\_xticks(df\_match\_summary["Fecha"])

ax1.set\_xticklabels(tick\_labels, rotation=0, ha="center")

# barras (eje derecho) -> usar matplotlib, no seaborn

ax2 = ax1.twinx()

ax2.bar(

    df\_match\_summary["Fecha"],

    df\_match\_summary["descanso\_prom"],

    width=4,          # ~4 días de ancho, ajustá a gusto

    alpha=0.25,

    align="center",

    edgecolor="none"

)

ax2.set\_ylabel("Días de descanso promedio")

# límites amables para ver bien las barras

if df\_match\_summary["descanso\_prom"].notna().any():

    ymax = float(df\_match\_summary["descanso\_prom"].max())

    ax2.set\_ylim(0, max(7, ymax \* 1.4))

plt.tight\_layout()

plt.show()

# --- opcional: mostrar la tabla resumen con rival ---

df\_match\_summary

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

|  | **Fecha** | **Rival** | **Rend\_Partido\_Prom** | **RvE\_prom** | **descanso\_prom** | **descanso\_min** |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 2025-07-19 | LEON | 71.064821 | 166.806967 | NaN | NaN |
| 1 | 2025-07-26 | SAN LUIS | 74.621851 | 184.176245 | 7.000000 | 7.0 |
| 2 | 2025-07-31 | NEW YORK RB | 73.886927 | 190.071523 | 5.000000 | 5.0 |
| 3 | 2025-08-03 | CHARLOTTE | 69.679054 | 171.017259 | 3.000000 | 3.0 |
| 4 | 2025-08-07 | CINCINNATI | 69.862237 | 198.235101 | 4.785714 | 4.0 |
| 5 | 2025-08-10 | SANTOS | 76.358902 | 207.507556 | 5.470588 | 3.0 |
| 6 | 2025-08-16 | C. JUAREZ | 58.832715 | 184.744529 | 6.352941 | 6.0 |
| 7 | 2025-08-22 | XOLOS | 73.325155 | 214.070270 | 6.352941 | 6.0 |
| 8 | 2025-08-30 | CRUZ AZUL | 69.320332 | 174.035263 | 9.400000 | 8.0 |

import matplotlib.dates as mdates

# --- Asegurar fechas ---

df\_part["Fecha"] = pd.to\_datetime(df\_part["Fecha"], errors="coerce")

df\_ent["Fecha"]  = pd.to\_datetime(df\_ent["Fecha"],  errors="coerce")

# --- Alinear tipos de id\_jugador (por si vienen como str) ---

for \_df in (df\_part, df\_ent):

    if "id\_jugador" in \_df.columns:

        \_df["id\_jugador"] = pd.to\_numeric(\_df["id\_jugador"], errors="coerce").astype("Int64")

# --- Helper de alias por nombre exacto o parecido (case-insensitive) ---

def pick(colnames, \*candidatos):

    low = {c.lower(): c for c in colnames}

    for c in candidatos:

        if c in colnames:                return c

        if c.lower() in low:             return low[c.lower()]

    return None

# Entrenos: mapeo a tus nombres

COL\_FECHA\_E  = pick(df\_ent.columns, "Fecha")

COL\_ID\_E     = pick(df\_ent.columns, "id\_jugador", "Id\_Jugador", "ID\_JUGADOR")

COL\_CE\_E     = pick(df\_ent.columns, "Carga\_Explosiva", "CE")

COL\_CS\_E     = pick(df\_ent.columns, "Carga\_Sosotenida", "Carga\_Sostenida", "CS")  # acepta el typo y el correcto

COL\_CR\_E     = pick(df\_ent.columns, "Carga\_Regenerativa", "CR")

COL\_REND\_E   = pick(df\_ent.columns, "Rendimiento\_Diario", "Rendimiento\_Partido")  # fallback si no está el diario

# Partidos

COL\_FECHA\_P  = pick(df\_part.columns, "Fecha")

COL\_ID\_P     = pick(df\_part.columns, "id\_jugador", "Id\_Jugador", "ID\_JUGADOR")

COL\_RIVAL\_P  = pick(df\_part.columns, "Rival")

COL\_REND\_P   = pick(df\_part.columns, "Rendimiento\_Partido")

COL\_RVE\_P    = pick(df\_part.columns, "Rendimiento\_vs\_Entreno")

# Sanity mínimo

needed = [COL\_FECHA\_E, COL\_ID\_E, COL\_CE\_E, COL\_CS\_E, COL\_CR\_E, COL\_FECHA\_P, COL\_ID\_P, COL\_RIVAL\_P, COL\_REND\_P]

assert all(x is not None for x in needed), "Faltan columnas necesarias; revisá nombres."

# --- Cruce por partido: carga de los 7 días previos de los jugadores que participaron ---

records = []

for fpart, g in df\_part.groupby(COL\_FECHA\_P):

    if pd.isna(fpart):

        continue

    f\_inicio = fpart - pd.Timedelta(days=7)

    jugadores = g[COL\_ID\_P].dropna().unique()

    riv = g[COL\_RIVAL\_P].dropna().mode().iloc[0] if not g[COL\_RIVAL\_P].dropna().empty else None

    df\_prev = df\_ent[

        (df\_ent[COL\_ID\_E].isin(jugadores)) &

        (df\_ent[COL\_FECHA\_E] >= f\_inicio) &

        (df\_ent[COL\_FECHA\_E] <  fpart)

    ]

    ce\_sum = df\_prev[COL\_CE\_E].sum() if not df\_prev.empty else np.nan

    cs\_sum = df\_prev[COL\_CS\_E].sum() if not df\_prev.empty else np.nan

    cr\_sum = df\_prev[COL\_CR\_E].sum() if not df\_prev.empty else np.nan

    rend\_ent\_prom = df\_prev[COL\_REND\_E].mean() if (COL\_REND\_E in df\_prev.columns and not df\_prev.empty) else np.nan

    records.append({

        "Fecha": fpart,

        "Rival": riv,

        "Rend\_Partido\_Prom": g[COL\_REND\_P].mean(),

        "RvE\_prom": g[COL\_RVE\_P].mean() if COL\_RVE\_P else np.nan,

        "CE\_prev7d": ce\_sum,

        "CS\_prev7d": cs\_sum,

        "CR\_prev7d": cr\_sum,

        "Rend\_Entreno\_Prev7d": rend\_ent\_prom

    })

df\_match\_training = pd.DataFrame(records).sort\_values("Fecha")

# --- Etiquetas eje X: Rival + Fecha (o dejá solo Rival si querés) ---

labels\_fecha\_rival = [f"{d.strftime('%Y-%m-%d')}\n{r}" for d, r in zip(df\_match\_training["Fecha"], df\_match\_training["Rival"])]

labels\_solo\_rival  = df\_match\_training["Rival"].tolist()  # mantiene el orden cronológico por estar sorteado por Fecha

# ===================== GRÁFICO COMBINADO =====================

fig, ax1 = plt.subplots(figsize=(12,6))

# --- Líneas: rendimiento partido y RvE ---

sns.lineplot(data=df\_match\_training, x="Fecha", y="Rend\_Partido\_Prom",

             marker="o", markersize=7, linewidth=2, label="Rend. Partido", ax=ax1)

sns.lineplot(data=df\_match\_training, x="Fecha", y="RvE\_prom",

             marker="s", markersize=7, linewidth=2, label="RvE prom", ax=ax1)

ax1.set\_ylabel("Rendimiento / RvE")

ax1.set\_xlabel("Fecha (debajo: Rival)")

ax1.set\_title("Rendimiento en partidos vs cargas previas (7 días)")

# Reemplazo ticks por etiqueta combinada Fecha+Rival

ax1.set\_xticks(df\_match\_training["Fecha"])

labels\_fecha\_rival = [f"{d.strftime('%Y-%m-%d')}\n{r}" for d, r in zip(df\_match\_training["Fecha"], df\_match\_training["Rival"])]

ax1.set\_xticklabels(labels\_fecha\_rival, rotation=0, ha="center")

# --- Barras: CE/CS/CR previos ---

ax2 = ax1.twinx()

x = mdates.date2num(df\_match\_training["Fecha"])

# ancho más chico

w = 1.5

ax2.bar(x - w, df\_match\_training["CE\_prev7d"], width=w, alpha=0.25, label="CE prev7d", color="blue")

ax2.bar(x,     df\_match\_training["CS\_prev7d"], width=w, alpha=0.25, label="CS prev7d", color="orange")

ax2.bar(x + w, df\_match\_training["CR\_prev7d"], width=w, alpha=0.25, label="CR prev7d", color="green")

ax2.set\_ylabel("Cargas previas (suma 7d)")

# --- Leyenda combinada ---

h1, l1 = ax1.get\_legend\_handles\_labels()

h2, l2 = ax2.get\_legend\_handles\_labels()

ax1.legend(h1+h2, l1+l2, loc="upper left", ncol=2)

plt.tight\_layout()

plt.show()

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

# Columnas necesarias

for c in ["Fecha","Rival","CE\_prev7d","CS\_prev7d","CR\_prev7d","Rend\_Partido\_Prom","RvE\_prom"]:

    assert c in df\_match\_training.columns, f"Falta la columna {c} en df\_match\_training"

dfm = df\_match\_training.sort\_values("Fecha").copy()

x = mdates.date2num(dfm["Fecha"])

labels = [f"{d.strftime('%Y-%m-%d')}\n{r}" for d, r in zip(dfm["Fecha"], dfm["Rival"])]

w = 1.4  # un poco más finas

fig, ax1 = plt.subplots(figsize=(12,6))

# ----- LÍNEAS (eje izquierdo) -----

ax1.plot(dfm["Fecha"], dfm["Rend\_Partido\_Prom"], marker="o", linewidth=2, label="Rend. Partido")

ax1.plot(dfm["Fecha"], dfm["RvE\_prom"],           marker="s", linewidth=2, label="RvE prom")

ax1.set\_ylabel("Rendimiento / RvE")

ax1.set\_xlabel("Fecha (debajo: Rival)")

ax1.set\_title("Rendimiento vs carga previa (7 días) — Barras apiladas")

# ticks personalizados

ax1.set\_xticks(dfm["Fecha"])

ax1.set\_xticklabels(labels, rotation=0, ha="center")

# ----- BARRAS APILADAS (eje derecho) -----

ax2 = ax1.twinx()

ce = dfm["CE\_prev7d"].fillna(0)

cs = dfm["CS\_prev7d"].fillna(0)

cr = dfm["CR\_prev7d"].fillna(0)

ax2.bar(x, ce, width=w, alpha=0.35, label="CE prev7d")

ax2.bar(x, cs, width=w, alpha=0.35, bottom=ce, label="CS prev7d")

ax2.bar(x, cr, width=w, alpha=0.35, bottom=(ce+cs), label="CR prev7d")

ax2.set\_ylabel("Cargas previas (suma 7d)")

# Leyenda combinada

h1,l1 = ax1.get\_legend\_handles\_labels()

h2,l2 = ax2.get\_legend\_handles\_labels()

ax1.legend(h1+h2, l1+l2, loc="upper left", ncol=3)

ax1.grid(axis="y", alpha=0.3)

plt.tight\_layout()

plt.show()

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

# ===== Preparación / sanity checks =====

import os

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Encontrar columna de nombre de jugador

col\_nombre = None

for c in ["nombre", "Nombre", "jugador", "Jugador"]:

    if c in df\_part.columns:

        col\_nombre = c

        break

if col\_nombre is None:

    raise ValueError("No encuentro columna de nombre de jugador (nombre/Nombre/jugador/Jugador).")

# Tipos y orden

if "Fecha" in df\_part.columns:

    df\_part["Fecha"] = pd.to\_datetime(df\_part["Fecha"], errors="coerce")

df\_part = df\_part.sort\_values(["id\_jugador", "Fecha"], kind="mergesort")

# Local/Visitante normalizado

if "local\_visitante" in df\_part.columns:

    df\_part["local\_visitante"] = df\_part["local\_visitante"].str.strip().str.lower().map({

        "l": "Local", "local": "Local",

        "v": "Visitante", "visitante": "Visitante",

    }).fillna(df\_part["local\_visitante"])

# RvE continuo (si no existe numérico)

if "Rendimiento\_vs\_Entreno" in df\_part.columns:

    df\_part["RvE\_pct"] = pd.to\_numeric(df\_part["Rendimiento\_vs\_Entreno"], errors="coerce")

else:

    # fallback si viene como RvE

    if "RvE" in df\_part.columns:

        df\_part["RvE\_pct"] = pd.to\_numeric(df\_part["RvE"], errors="coerce")

    else:

        df\_part["RvE\_pct"] = np.nan

# Flag categórico consistente

if "RvE\_flag" not in df\_part.columns:

    def \_cat\_rve(x):

        if pd.isna(x): return "Sin base"

        if x >= 250:   return "Alto (≥250%)"

        if x >= 150:   return "Moderado (150–249%)"

        return "OK (<150%)"

    df\_part["RvE\_flag"] = df\_part["RvE\_pct"].apply(\_cat\_rve)

# Carpeta de export

export\_dir = "exports\_partidos"

os.makedirs(export\_dir, exist\_ok=True)

# Paletas y estilos

sns.set(style="whitegrid", context="notebook")

# Ordenar rivales por frecuencia

riv\_freq = df\_part["Rival"].value\_counts().index.tolist()

pivot\_rve = (df\_part

             .groupby(["Rival", "linea"], as\_index=False)["RvE\_pct"]

             .mean())

pivot\_cnt = (df\_part

             .groupby(["Rival", "linea"], as\_index=False)["RvE\_pct"]

             .size()

             .rename(columns={"size": "n"}))

heat = pivot\_rve.pivot\_table(index="Rival", columns="linea", values="RvE\_pct")

heat = heat.reindex(riv\_freq)  # por frecuencia

plt.figure(figsize=(10, max(6, 0.4\*len(heat))))

sns.heatmap(heat, annot=True, fmt=".0f", linewidths=.5, cbar\_kws={"label": "RvE (%)"})

plt.title("Matriz Rival × Línea — RvE promedio")

plt.xlabel("Línea")

plt.ylabel("Rival")

plt.tight\_layout()

plt.show()

# Export tabla con conteos también

tbl\_rival = (pivot\_rve.merge(pivot\_cnt, on=["Rival","linea"], how="left")

             .sort\_values(["Rival","linea"]))

tbl\_rival.to\_csv(f"{export\_dir}/rival\_linea\_rve\_promedio\_conteo.csv", index=False)

tbl\_rival.head(10)

Gráfico

El contenido generado por IA puede ser incorrecto.

# Ordenar rivales por frecuencia

riv\_freq = df\_part["Rival"].value\_counts().index.tolist()

pivot\_rve = (df\_part

             .groupby(["Rival", "nombre"], as\_index=False)["RvE\_pct"]

             .mean())

pivot\_cnt = (df\_part

             .groupby(["Rival", "nombre"], as\_index=False)["RvE\_pct"]

             .size()

             .rename(columns={"size": "n"}))

heat = pivot\_rve.pivot\_table(index="Rival", columns="nombre", values="RvE\_pct")

heat = heat.reindex(riv\_freq)  # por frecuencia

plt.figure(figsize=(10, max(6, 0.4\*len(heat))))

sns.heatmap(heat, annot=True, fmt=".0f", linewidths=.5, cbar\_kws={"label": "RvE (%)"})

plt.title("Matriz Rival × Jugador — RvE promedio")

plt.xlabel("Nombre")

plt.ylabel("Rival")

plt.tight\_layout()

plt.show()

# Export tabla con conteos también

tbl\_rival = (pivot\_rve.merge(pivot\_cnt, on=["Rival","nombre"], how="left")

             .sort\_values(["Rival","nombre"]))

tbl\_rival.to\_csv(f"{export\_dir}/rival\_linea\_rve\_promedio\_conteo.csv", index=False)

tbl\_rival.head(10)

Escala de tiempo

El contenido generado por IA puede ser incorrecto.

# Promedios por Local/Visitante y línea

lv\_linea = (df\_part

            .dropna(subset=["Local\_Visitante"])

            .groupby(["Local\_Visitante", "linea"], as\_index=False)["RvE\_pct"]

            .mean()

            .rename(columns={"RvE\_pct":"RvE\_prom"}))

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

sns.barplot(data=lv\_linea, x="linea", y="RvE\_prom", hue="Local\_Visitante")

plt.axhline(150, linestyle="--", linewidth=1)

plt.axhline(250, linestyle="--", linewidth=1)

plt.title("RvE promedio por línea y condición de localía")

plt.ylabel("RvE (%) promedio")

plt.xlabel("")

plt.legend(title="Condición")

plt.show()

# Global

lv\_global = (df\_part

             .dropna(subset=["Local\_Visitante"])

             .groupby("Local\_Visitante", as\_index=False)["RvE\_pct"]

             .mean()

             .rename(columns={"RvE\_pct":"RvE\_prom"}))

lv\_global.to\_csv(f"{export\_dir}/local\_visitante\_global.csv", index=False)

lv\_global

# Promedios por Local/Visitante y línea

lv\_linea = (df\_part

            .dropna(subset=["Local\_Visitante"])

            .groupby(["Local\_Visitante", "linea"], as\_index=False)["RvE\_pct"]

            .mean()

            .rename(columns={"RvE\_pct":"RvE\_prom"}))

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

sns.barplot(data=lv\_linea, x="linea", y="RvE\_prom", hue="Local\_Visitante")

plt.axhline(150, linestyle="--", linewidth=1)

plt.axhline(250, linestyle="--", linewidth=1)

plt.title("RvE promedio por línea y condición de localía")

plt.ylabel("RvE (%) promedio")

plt.xlabel("")

plt.legend(title="Condición")

plt.show()

# Global

lv\_global = (df\_part

             .dropna(subset=["Local\_Visitante"])

             .groupby("Local\_Visitante", as\_index=False)["RvE\_pct"]

             .mean()

             .rename(columns={"RvE\_pct":"RvE\_prom"}))

lv\_global.to\_csv(f"{export\_dir}/local\_visitante\_global.csv", index=False)

lv\_global

Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.

| **Local\_Visitante** | **RvE\_prom** |
| --- | --- |
| 0 | Local | 182.746303 |
| 1 | Visitante | 191.499425 |

# Rolling por jugador (suaviza ruido partido a partido)

win = 3  # partidos

df\_part["\_rve\_roll"] = (df\_part

                        .sort\_values(["id\_jugador","Fecha"])

                        .groupby("id\_jugador")["RvE\_pct"]

                        .transform(lambda s: s.rolling(win, min\_periods=1).mean()))

# Gráfico para un jugador específico (cambiá el nombre)

jugador\_demo = df\_part[col\_nombre].value\_counts().index[0]

tmp = df\_part[df\_part[col\_nombre]==jugador\_demo]

plt.figure(figsize=(10,4))

plt.plot(tmp["Rival"], tmp["RvE\_pct"], marker="o", linewidth=1, label="RvE (%)")

plt.plot(tmp["Rival"], tmp["\_rve\_roll"], linewidth=2, label=f"Media móvil ({win})")

plt.axhline(150, linestyle="--", linewidth=1)

plt.axhline(250, linestyle="--", linewidth=1)

plt.title(f"Tendencia RvE — {jugador\_demo}")

plt.ylabel("RvE (%)")

plt.xlabel("Rival")

plt.legend()

plt.tight\_layout()

plt.show()

# Top / Bottom partidos (por jugador)

def top\_bottom\_por\_jugador(df, k=3):

    rows = []

    for jid, g in df.groupby("id\_jugador"):

        nombre = g[col\_nombre].iloc[0]

        gb = g.dropna(subset=["RvE\_pct"]).copy()

        if gb.empty:

            continue

        tb\_top = gb.nlargest(k, "RvE\_pct")[["Fecha","Rival","Local\_Visitante","RvE\_pct", col\_nombre]]

        tb\_top["tipo"] = "Top"

        tb\_bot = gb.nsmallest(k, "RvE\_pct")[["Fecha","Rival","Local\_Visitante","RvE\_pct", col\_nombre]]

        tb\_bot["tipo"] = "Bottom"

        out = pd.concat([tb\_top, tb\_bot], ignore\_index=True)

        out.insert(0, "id\_jugador", jid)

        rows.append(out)

    return pd.concat(rows, ignore\_index=True) if rows else pd.DataFrame()

tb = top\_bottom\_por\_jugador(df\_part, k=3)

tb = tb.sort\_values([col\_nombre, "tipo", "RvE\_pct"], ascending=[True, True, False])

tb.to\_csv(f"{export\_dir}/top\_bottom\_rve\_por\_jugador.csv", index=False)

tb.head(12)

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

|  | **id\_jugador** | **Fecha** | **Rival** | **Local\_Visitante** | **RvE\_pct** | **nombre** | **tipo** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 11 | 4 | 2025-07-26 | SAN LUIS | Local | 147.181606 | Alan Mozo | Bottom |
| 10 | 4 | 2025-07-31 | NEW YORK RB | Visitante | 103.215798 | Alan Mozo | Bottom |
| 9 | 4 | 2025-08-03 | CHARLOTTE | Visitante | 100.054504 | Alan Mozo | Bottom |
| 6 | 4 | 2025-08-07 | CINCINNATI | Visitante | 250.000000 | Alan Mozo | Top |
| 7 | 4 | 2025-08-10 | SANTOS | Local | 250.000000 | Alan Mozo | Top |
| 8 | 4 | 2025-08-16 | C. JUAREZ | Local | 250.000000 | Alan Mozo | Top |
| 104 | 23 | 2025-07-19 | LEON | Visitante | 250.000000 | Alan Pulido | Bottom |
| 105 | 23 | 2025-07-26 | SAN LUIS | Local | 250.000000 | Alan Pulido | Bottom |
| 103 | 23 | 2025-08-16 | C. JUAREZ | Local | 70.732691 | Alan Pulido | Bottom |
| 100 | 23 | 2025-07-19 | LEON | Visitante | 250.000000 | Alan Pulido | Top |
| 101 | 23 | 2025-07-26 | SAN LUIS | Local | 250.000000 | Alan Pulido | Top |
| 102 | 23 | 2025-07-31 | NEW YORK RB | Visitante | 250.000000 | Alan Pulido | Top |

# Conteo de flags por jugador

flags\_jugador = (df\_part

                 .groupby([col\_nombre, "RvE\_flag"], as\_index=False)

                 .size()

                 .rename(columns={"size":"n"}))

pivot\_flags\_j = flags\_jugador.pivot(index=col\_nombre, columns="RvE\_flag", values="n").fillna(0).astype(int)

pivot\_flags\_j["Total"] = pivot\_flags\_j.sum(axis=1)

pivot\_flags\_j = pivot\_flags\_j.sort\_values("Total", ascending=False)

pivot\_flags\_j.to\_csv(f"{export\_dir}/resumen\_flags\_por\_jugador.csv")

pivot\_flags\_j.head(10)

# Conteo de flags por línea

flags\_linea = (df\_part

               .groupby(["linea", "RvE\_flag"], as\_index=False)

               .size()

               .rename(columns={"size":"n"}))

pivot\_flags\_l = flags\_linea.pivot(index="linea", columns="RvE\_flag", values="n").fillna(0).astype(int)

pivot\_flags\_l["Total"] = pivot\_flags\_l.sum(axis=1)

pivot\_flags\_l.to\_csv(f"{export\_dir}/resumen\_flags\_por\_linea.csv")

pivot\_flags\_l

| **RvE\_flag** | **<50% (muy por debajo)** | **Normal** | **Total** |
| --- | --- | --- | --- |
| **linea** |  |  |  |
| Defensa Central | 1 | 19 | 20 |
| Defensa Lateral | 3 | 25 | 28 |
| Delantera | 3 | 17 | 20 |
| Extremo | 6 | 35 | 41 |
| Medio Defensivo | 1 | 17 | 18 |
| Medio Ofensivo | 3 | 14 | 17 |
| Portería | 0 | 4 | 4 |

# Partidos con RvE muy alto o muy bajo (posibles issues de baseline o cargas)

thr\_high = 300

thr\_low  = 50

outliers = df\_part.loc[(df\_part["RvE\_pct"]>=thr\_high) | (df\_part["RvE\_pct"]<=thr\_low),

                       ["Fecha", col\_nombre, "linea", "Rival", "Local\_Visitante", "RvE\_pct", "Rendimiento\_Partido"]].copy()

outliers = outliers.sort\_values("RvE\_pct", ascending=False)

outliers.to\_csv(f"{export\_dir}/partidos\_outliers\_rve.csv", index=False)

outliers.head(15)

| **Fecha** | **nombre** | **linea** | **Rival** | **Local\_Visitante** | **RvE\_pct** | **Rendimiento\_Partido** |
| --- | --- | --- | --- | --- | --- | --- |
| 104 | 2025-08-16 | Hugo Camberos | Extremo | C. JUAREZ | Local | 48.694731 | 16.775681 |
| 144 | 2025-08-30 | Miguel Gómez | Defensa Lateral | CRUZ AZUL | Local | 41.433342 | 15.932817 |
| 17 | 2025-07-26 | Armando González | Delantera | SAN LUIS | Local | 41.221073 | 19.426370 |
| 96 | 2025-08-16 | Santiago Sandoval | Medio Ofensivo | C. JUAREZ | Local | 33.164132 | 23.301412 |
| 88 | 2025-07-31 | Rubén González | Medio Defensivo | NEW YORK RB | Visitante | 27.496113 | 7.779231 |
| 130 | 2025-08-16 | Raúl Martínez | Defensa Central | C. JUAREZ | Local | 23.941922 | 10.956138 |
| 81 | 2025-08-03 | Miguel Gómez | Defensa Lateral | CHARLOTTE | Visitante | 18.476550 | 5.552303 |
| 61 | 2025-07-26 | Hugo Camberos | Extremo | SAN LUIS | Local | 16.257320 | 6.430394 |
| 147 | 2025-08-30 | Yael Padilla | Medio Ofensivo | CRUZ AZUL | Local | 12.699350 | 6.091018 |
| 91 | 2025-08-07 | Teun Wilke | Delantera | CINCINNATI | Visitante | 10.128653 | 3.168765 |
| 32 | 2025-08-10 | Cade Cowell | Extremo | SANTOS | Visitante | 8.516063 | 1.773823 |
| 18 | 2025-07-19 | Armando González | Delantera | LEON | Visitante | 5.819171 | 2.005272 |
| 98 | 2025-08-16 | Miguel Gómez | Defensa Lateral | C. JUAREZ | Local | 4.506945 | 1.708203 |
| 27 | 2025-08-03 | Cade Cowell | Extremo | CHARLOTTE | Visitante | 0.000000 | 0.000000 |
| 141 | 2025-08-30 | Hugo Camberos | Extremo | CRUZ AZUL | Local | 0.000000 | 0.000000 |

# KPIs por jugador: promedio RvE, % partidos ≥150, ≥250, partidos, y promedio Rendimiento\_Partido

def kpis\_jugador(df):

    g = df.groupby("id\_jugador", as\_index=False).agg(

        partidos=("RvE\_pct", "count"),

        rve\_prom=("RvE\_pct", "mean"),

        rve\_p90=("RvE\_pct", lambda s: np.nanpercentile(s.dropna(), 90) if s.notna().any() else np.nan),

        rve\_ge150=("RvE\_pct", lambda s: (s>=150).mean() if s.notna().any() else np.nan),

        rve\_ge250=("RvE\_pct", lambda s: (s>=250).mean() if s.notna().any() else np.nan),

        rend\_part\_prom=("Rendimiento\_Partido", "mean"),

    )

    # Nombre/posición/línea

    meta = df.groupby("id\_jugador")[[col\_nombre,  "linea"]].agg(lambda s: s.dropna().iloc[0] if s.dropna().any() else np.nan)

    meta = meta.reset\_index()

    out = meta.merge(g, on="id\_jugador", how="right")

    out["rve\_ge150"] = (out["rve\_ge150"]\*100).round(1)

    out["rve\_ge250"] = (out["rve\_ge250"]\*100).round(1)

    return out.sort\_values(["rve\_prom","rve\_p90","partidos"], ascending=[False, False, False])

kpis = kpis\_jugador(df\_part)

kpis.to\_csv(f"{export\_dir}/kpis\_partidos\_por\_jugador.csv", index=False)

kpis

| **id\_jugador** | **nombre** | **linea** | **partidos** | **rve\_prom** | **rve\_p90** | **rve\_ge150** | **rve\_ge250** | **rend\_part\_prom** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 14 | 18 | Luis Romo | Medio Ofensivo | 9 | 250.000000 | 250.000000 | 100.0 | 100.0 | 96.331706 |
| 15 | 19 | Daniel Aguirre | Medio Defensivo | 7 | 250.000000 | 250.000000 | 100.0 | 100.0 | 81.647483 |
| 3 | 6 | Miguel Tapias | Defensa Central | 2 | 250.000000 | 250.000000 | 100.0 | 100.0 | 90.076135 |
| 10 | 14 | Omar Govea | Medio Defensivo | 2 | 250.000000 | 250.000000 | 100.0 | 100.0 | 89.517437 |
| 12 | 16 | Isaác Brisuela | Extremo | 2 | 250.000000 | 250.000000 | 100.0 | 100.0 | 66.979613 |
| 2 | 5 | Gilberto Sepulveda | Defensa Central | 6 | 242.727235 | 250.000000 | 100.0 | 66.7 | 96.175128 |
| 7 | 10 | José Castillo | Defensa Lateral | 7 | 236.844688 | 250.000000 | 100.0 | 71.4 | 88.261061 |
| 13 | 17 | Erick Gutiérrez | Medio Defensivo | 7 | 227.833864 | 250.000000 | 85.7 | 71.4 | 85.061310 |
| 19 | 23 | Alan Pulido | Delantera | 6 | 220.122115 | 250.000000 | 83.3 | 83.3 | 76.645666 |
| 21 | 25 | Roberto Alvarado | Extremo | 8 | 213.686589 | 250.000000 | 100.0 | 25.0 | 95.849774 |
| 11 | 15 | Efraín Álvarez | Extremo | 9 | 210.037710 | 250.000000 | 88.9 | 44.4 | 83.442536 |
| 4 | 7 | Bryan González | Defensa Lateral | 9 | 209.176026 | 250.000000 | 100.0 | 22.2 | 96.188911 |
| 17 | 21 | Richard Ledezma | Extremo | 7 | 208.240017 | 250.000000 | 85.7 | 57.1 | 81.477671 |
| 20 | 24 | Javier Hernández | Delantera | 3 | 201.743844 | 250.000000 | 66.7 | 66.7 | 35.999445 |
| 6 | 9 | Diego Campillo | Defensa Central | 9 | 199.361227 | 250.000000 | 77.8 | 33.3 | 78.997319 |
| 1 | 4 | Alan Mozo | Defensa Lateral | 7 | 192.921701 | 250.000000 | 57.1 | 57.1 | 66.850432 |
| 5 | 8 | Luis Olivas | Defensa Central | 2 | 179.023308 | 190.359575 | 100.0 | 0.0 | 63.724294 |
| 25 | 29 | Armando González | Delantera | 9 | 168.664356 | 250.000000 | 66.7 | 55.6 | 64.950417 |
| 0 | 1 | Raúl Rangel | Portería | 4 | 142.384860 | 152.724316 | 25.0 | 0.0 | 19.277704 |
| 16 | 20 | Rubén González | Medio Defensivo | 2 | 138.748056 | 227.749611 | 50.0 | 50.0 | 42.909338 |
| 18 | 22 | Cade Cowell | Extremo | 9 | 129.350930 | 250.000000 | 33.3 | 22.2 | 38.887653 |
| 23 | 27 | Teun Wilke | Delantera | 2 | 125.607072 | 217.989806 | 50.0 | 0.0 | 36.215268 |
| 24 | 28 | Yael Padilla | Medio Ofensivo | 3 | 121.775986 | 220.525722 | 33.3 | 33.3 | 48.331002 |
| 26 | 31 | Santiago Sandoval | Medio Ofensivo | 5 | 90.829738 | 152.579869 | 40.0 | 0.0 | 62.668504 |
| 22 | 26 | Hugo Camberos | Extremo | 6 | 80.308016 | 158.926857 | 16.7 | 0.0 | 28.681510 |
| 8 | 11 | Miguel Gómez | Defensa Lateral | 5 | 40.635026 | 70.102854 | 0.0 | 0.0 | 12.441180 |
| 9 | 13 | Raúl Martínez | Defensa Central | 1 | 23.941922 | 23.941922 | 0.0 | 0.0 | 10.956138 |