Carga de datos

```
        # -------------------------
# 1) CARGA (usa tu query original)
# -------------------------
base = spark.sql("""select
cosecha,codmes,maduracion,saldo_inicial,pdponderado,rango_pd,porcentajeacum as
provisionporcacum
           from catalog_lhcl_prod_bcp.bcp_edv_rbm.T65734_caldesagregada where
cosecha>=202201""").toPandas()

base = base.rename(columns={
    "Maduracion": "maduracion",
    "PD_Ponderado": "pdponderado",
    "Provision": "provisionporcacum"
})
```

```
base["maduracion"] = base["maduracion"].astype(float)
base["pdponderado"] = base["pdponderado"].astype(float)
base["provisionporcacum"] = base["provisionporcacum"].astype(float)
data = base[(base["maduracion"] >= 2)#& (base["maduracion"] <= 16)
            #&(base["cosecha"].isin([202411,202412,202410,202407,202409,202408,202406,
202405,202401]))
            &(base["cosecha"]>202400)].copy()
```

Logístico

```
base["maduracion"] = base["maduracion"].astype(float)
base["pdponderado"] = base["pdponderado"].astype(float)
base["provisionporcacum"] = base["provisionporcacum"].astype(float)
data = base[(base["maduracion"] >= 2)#& (base["maduracion"] <= 16)
            #&(base["cosecha"].isin([202411,202412,202410,202407,202409,202408,202406,
202405,202401]))
            &(base["cosecha"]>202400)].copy()
```

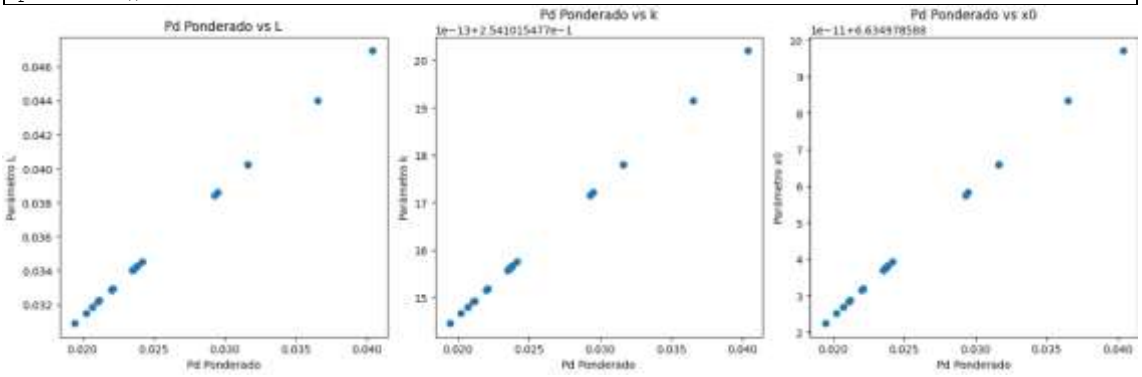
```
###para encontrar una relación entre las variables de cada modelo logístico
from scipy.optimize import least_squares

# === 1. Preparar datos ===
# data: DataFrame con columnas
['cosecha','maduracion','provisionporcacum','pdponderado']
# Asumimos que ya lo tienes cargado

def modelo_logistico(x, L, k, x0):
    return L / (1 + np.exp(-k * (x - x0)))

# Armamos matrices por fila (cada registro)
x = data['maduracion'].values.astype(float)
y = data['provisionporcacum'].values.astype(float)
pdw = data['pdponderado'].values.astype(float)

# === 2. Definir parámetros globales como función de pdponderado ===
# p = [a0, a1, b0, b1, c0, c1]
def residuals(p, x, y, pdw):
    a0, a1, b0, b1, c0, c1 = p
    L = a0 + a1 * pdw
    k = b0 + b1 * pdw
    x0 = c0 + c1 * pdw

    # Evitar negativos numéricamente: recortamos una mínima cota
    L = np.clip(L, 1e-9, None)
    k = np.clip(k, 1e-9, None)

    yhat = modelo_logistico(x, L, k, x0)
    return yhat - y

# === 3. Inicialización razonable (usa tus escalas) ===
p0 = np.array([
    np.median(y),      # a0 ~ nivel base de L
    1.0,               # a1: impacto de pd en L
    0.1,               # b0 ~ base de k
    0.5,               # b1: impacto de pd en k
    np.median(x),      # c0 ~ base de x0
```

```
    5.0               # c1: impacto de pd en x0
], dtype=float)

# === 4. Límites para asegurar monotonía y rangos razonables ===
# a1, b1, c1 >= 0  (monotonicidad creciente)
lower_bounds = np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])   # evita negativos
upper_bounds = np.array([1.0,  5.0, 1.0,  5.0, 40.0, 40.0]) # ajusta según tus escalas

res = least_squares(residuals, p0, bounds=(lower_bounds, upper_bounds), args=(x, y,
pdw), max_nfev=20000)

p_opt = res.x
a0, a1, b0, b1, c0, c1 = p_opt

# === 5. Construir tabla por cosecha con parámetros inducidos por su pdponderado ===
tabla_coef = (
    data.groupby('cosecha')['pdponderado']
        .mean()
        .reset_index()
        .rename(columns={'pdponderado':'pdponderado_mean'})
)

tabla_coef['L']  = a0 + a1 * tabla_coef['pdponderado_mean']
tabla_coef['k']  = b0 + b1 * tabla_coef['pdponderado_mean']
tabla_coef['x0'] = c0 + c1 * tabla_coef['pdponderado_mean']

# Opcional: recortar mínimos
tabla_coef['L']  = tabla_coef['L'].clip(lower=1e-9)
tabla_coef['k']  = tabla_coef['k'].clip(lower=1e-9)

print(tabla_coef)
```

	cosecha	pdponderado_mean	L	k	x0
0	202401	0.040372	0.04694	0.254102	6.634979
1	202402	0.036526	0.043988	0.254102	6.634979
2	202403	0.031617	0.040221	0.254102	6.634979
3	202404	0.031613	0.040218	0.254102	6.634979
4	202405	0.029489	0.038588	0.254102	6.634979
5	202406	0.029286	0.038432	0.254102	6.634979
6	202407	0.023694	0.03414	0.254102	6.634979
7	202408	0.024186	0.034518	0.254102	6.634979
8	202409	0.023737	0.034173	0.254102	6.634979
9	202410	0.023531	0.034015	0.254102	6.634979
10	202411	0.021203	0.032229	0.254102	6.634979
11	202412	0.022025	0.03286	0.254102	6.634979
12	202501	0.020711	0.031851	0.254102	6.634979
13	202502	0.022133	0.032942	0.254102	6.634979
14	202503	0.023521	0.034008	0.254102	6.634979
15	202504	0.023864	0.034271	0.254102	6.634979
16	202505	0.019455	0.030887	0.254102	6.634979
17	202506	0.020243	0.031492	0.254102	6.634979
18	202507	0.021092	0.032144	0.254102	6.634979
19	202508	0.022046	0.032876	0.254102	6.634979

```
filtered_df = tabla_coef
# ============================
# 7. Scatter de parámetros vs pdponderado
# ============================
filtered_df = tabla_coef #[tabla_coef["cosecha"] > 202400]

fig, axs = plt.subplots(1, 3, figsize=(15, 5))

axs[0].scatter(filtered_df['pdponderado_mean'], filtered_df['L'])
axs[0].set_xlabel('Pd Ponderado')
```

```
axs[0].set_ylabel('Parámetro L') # límite superior: provisiónacum no puede crecer
infinito
axs[0].set_title('Pd Ponderado vs L')

axs[1].scatter(filtered_df['pdponderado_mean'], filtered_df['k'])
axs[1].set_xlabel('Pd Ponderado')
axs[1].set_ylabel('Parámetro k') #parametro de crecimiento
axs[1].set_title('Pd Ponderado vs k')

axs[2].scatter(filtered_df['pdponderado_mean'], filtered_df['x0'])
axs[2].set_xlabel('Pd Ponderado')
axs[2].set_ylabel('Parámetro x0')
axs[2].set_title('Pd Ponderado vs x0')

plt.tight_layout()
plt.show()
```




```
# Supongamos que ya tienes los coeficientes del modelo A en p_opt:
# p_opt = [a0, a1, b0, b1, c0, c1]
a0, a1, b0, b1, c0, c1 = p_opt

# 1. Imprimir la fórmula en formato legible
print("Modelo logístico dependiente de pdponderado:")
print(f"L(pd)  = {a0:.6f} + {a1:.6f} * pdponderado")
print(f"k(pd)  = {b0:.6f} + {b1:.6f} * pdponderado")
print(f"x0(pd) = {c0:.6f} + {c1:.6f} * pdponderado")

print("\nFórmula completa:")
print(f"y(x,pd) = ( {a0:.6f} + {a1:.6f}*pd ) / ( 1 + exp( - ( {b0:.6f} + {b1:.6f}*pd )
* ( x - ( {c0:.6f} + {c1:.6f}*pd ) ) ) )")

# 2. Crear función Python para predecir
import numpy as np

def modelo_logistico_pd(x, pd):
    L = a0 + a1 * pd
    k = b0 + b1 * pd
    x0 = c0 + c1 * pd
    return L / (1 + np.exp(-k * (x - x0)))
```

Modelo logístico dependiente de pdponderado:

L(pd) = 0.015956 + 0.767455 * pdponderado

k(pd) = 0.254102 + 0.000000 * pdponderado

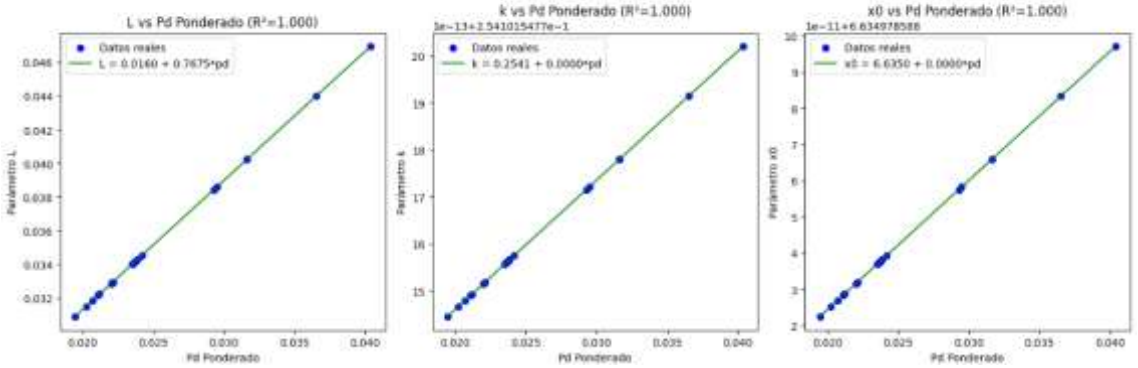
x0(pd) = 6.634979 + 0.000000 * pdponderado

Fórmula completa: y(x,pd) = (0.015956 + 0.767455*pd) / (1 + exp(- (0.254102 + 0.000000*pd) * (x - (6.634979 + 0.000000*pd))))

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import r2_score

# Coeficientes del modelo A (ya calculados)
a0, a1, b0, b1, c0, c1 = p_opt

# Datos reales
x_pd = tabla_coef['pdponderado_mean'].values
L_real = tabla_coef['L'].values
```

```
k_real = tabla_coef['k'].values
x0_real = tabla_coef['x0'].values

# Rango para la línea
pd_range = np.linspace(x_pd.min(), x_pd.max(), 100)

# Líneas ajustadas
L_line = a0 + a1 * pd_range
k_line = b0 + b1 * pd_range
x0_line = c0 + c1 * pd_range

from sklearn.metrics import r2_score
# Calcular predicciones para los puntos reales
L_pred = a0 + a1 * x_pd
k_pred = b0 + b1 * x_pd
x0_pred = c0 + c1 * x_pd

# Calcular R²
r2_L = r2_score(L_real, L_pred)
r2_k = r2_score(k_real, k_pred)
r2_x0 = r2_score(x0_real, x0_pred)

print(f"R² para L: {r2_L:.4f}")
print(f"R² para k: {r2_k:.4f}")
print(f"R² para x0: {r2_x0:.4f}")

# Agregar R² en los títulos del gráfico
fig, axs = plt.subplots(1, 3, figsize=(15, 5))

axs[0].scatter(x_pd, L_real, color='blue', label='Datos reales')
axs[0].plot(pd_range, L_line, color='green', label=f"L = {a0:.4f} + {a1:.4f}*pd")
axs[0].set_xlabel('Pd Ponderado')
axs[0].set_ylabel('Parámetro L')
axs[0].set_title(f'L vs Pd Ponderado (R²={r2_L:.3f})')
axs[0].legend()

axs[1].scatter(x_pd, k_real, color='blue', label='Datos reales')
axs[1].plot(pd_range, k_line, color='green', label=f"k = {b0:.4f} + {b1:.4f}*pd")
axs[1].set_xlabel('Pd Ponderado')
axs[1].set_ylabel('Parámetro k')
axs[1].set_title(f'k vs Pd Ponderado (R²={r2_k:.3f})')
axs[1].legend()

axs[2].scatter(x_pd, x0_real, color='blue', label='Datos reales')
axs[2].plot(pd_range, x0_line, color='green', label=f"x0 = {c0:.4f} + {c1:.4f}*pd")
axs[2].set_xlabel('Pd Ponderado')
axs[2].set_ylabel('Parámetro x0')
axs[2].set_title(f'x0 vs Pd Ponderado (R²={r2_x0:.3f})')
axs[2].legend()

plt.tight_layout()
plt.show()
```

```
R² para L: 1.0000
R² para k: 1.0000
R² para x0: 1.0000
```




```
a0, a1,b0,b1,c0,c1
```

(0.015956384261427606, 0.7674546103886146, 0.25410154775090943, 2.751359613085519e-11, 6.634978587552972, 3.569352350271897e-09)

Aplicando la fórmula

```
dataparte1=base[(base["maduracion"] >= 2)&((base["cosecha"] > 202400)
|(base["cosecha"] == 202301))].copy()

import plotly.graph_objects as go
import numpy as np

# Función del modelo logístico dependiente de pdponderado
def modelo_logistico_pd(x, pd):
    L = a0 + a1 * pd
    k = b0 + b1 * pd
    x0 = c0 + c1 * pd
    return L / (1 + np.exp(-k * (x - x0)))

# Crear figura
fig = go.Figure()

# Iterar por cada cosecha en dataparte1
for cosecha, grupo in dataparte1.groupby('cosecha'):
    x_real = grupo['maduracion'].values
    y_real = grupo['provisionporcacum'].values
    pd_val = grupo['pdponderado'].mean()  # promedio para el ajuste

    # Curva ajustada
    x_range = np.linspace(0, 40, 200)
#    x_range = np.linspace(x_real.min(), x_real.max(), 100)
    y_pred = modelo_logistico_pd(x_range, pd_val)

    # Agregar puntos reales
    fig.add_trace(go.Scatter(
        x=x_real, y=y_real,
        mode='markers+lines',
        name=f'{cosecha}, Real',
        line=dict(dash='solid'),
        marker=dict(size=6)
    ))

    # Agregar curva ajustada
    fig.add_trace(go.Scatter(
        x=x_range, y=y_pred,
        mode='lines',
        name=f'{cosecha}, Ajuste',
        line=dict(dash='dot')
    ))

# Layout
fig.update_layout(
    title='Curvas Logísticas Ajustadas vs Datos Reales',
    xaxis_title='Maduración',
    yaxis_title='ProvisiónPorcAcum',
    template='plotly_white',
    legend_title='Cosecha, Tipo'
)

fig.show()
```

Evaluación del r2

```
## EVALUACION DEL R^2
from sklearn.metrics import r2_score
import pandas as pd

r2_por_cosecha = {}

impacto_real_total = []
impacto_pred_total = []

for cosecha, grupo in dataparte1.groupby('cosecha'):
    x_real = grupo['maduracion'].values
    y_real = grupo['provisionporcacum'].values
    saldo = grupo['saldo_inicial'].mean()
    pd_val = grupo['pdponderado'].mean()

    # Impacto real
    impacto_real = y_real * saldo

    # Predicciones en el mismo rango que los datos reales
    y_pred_real_range = modelo_logistico_pd(x_real, pd_val)
    impacto_pred = y_pred_real_range * saldo

    # Calcular R² para esta cosecha
    r2 = r2_score(impacto_real, impacto_pred)
    r2_por_cosecha[cosecha] = r2

    # Acumular para R² global
    impacto_real_total.extend(impacto_real)
    impacto_pred_total.extend(impacto_pred)

# R² global
r2_global = r2_score(impacto_real_total, impacto_pred_total)

# Mostrar resultados
print("R² por cosecha:")
for cosecha, r2 in r2_por_cosecha.items():
    print(f"Cosecha {cosecha}: R² = {r2:.4f}")

print(f"\nR² global: {r2_global:.4f}")

# Opcional: convertir a DataFrame para análisis
r2_df = pd.DataFrame(list(r2_por_cosecha.items()), columns=['Cosecha', 'R²'])
print(r2_df)
```

R² global: 0.8631

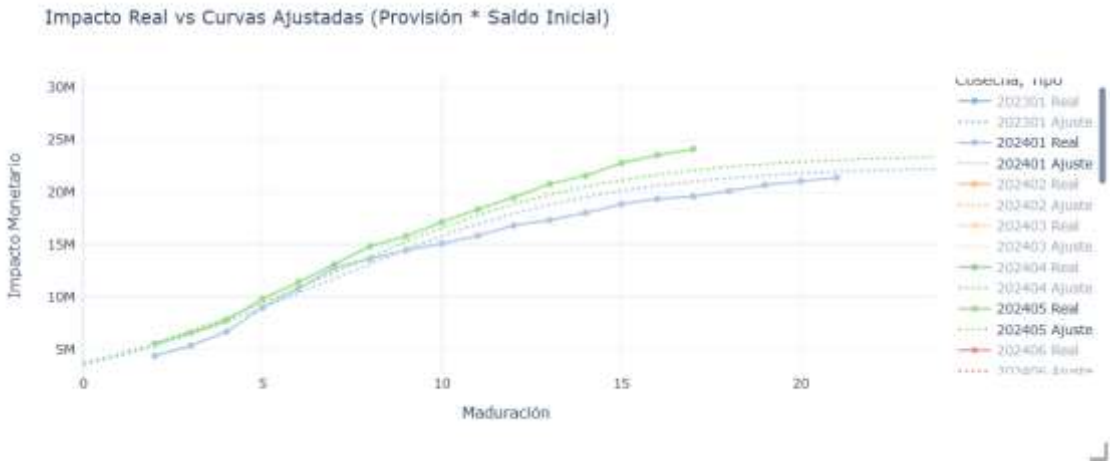
	Cosecha	R²
0	202301	0.199055
1	202401	0.962429
2	202402	0.93591

3	202403	0.888358
4	202404	0.581323
5	202405	0.972682
6	202406	0.898728
7	202407	0.712364
8	202408	0.994872
9	202409	0.960805
10	202410	0.907134
11	202411	0.931378
12	202412	0.969034
13	202501	0.963025
14	202502	0.29365
15	202503	0.973843
16	202504	0.95241
17	202505	0.883074

Proyección: Impacto en soles en cada cosecha

```
import plotly.graph_objects as go
import numpy as np
import matplotlib.cm as cm

# Función del modelo logístico dependiente de pdponderado
def modelo_logistico_pd(x, pd):
    L = a0 + a1 * pd
    k = b0 + b1 * pd
    x0 = c0 + c1 * pd
    return L / (1 + np.exp(-k * (x - x0)))

# Crear figura
fig = go.Figure()

# Lista de cosechas únicas
cosechas = dataparte1['cosecha'].unique()

# Generar paleta de colores (usamos tab20 para variedad)
colors = cm.get_cmap('tab20', len(cosechas))
color_map = {
    cosecha: f'rgba({int(colors(i)[0]*255)}, {int(colors(i)[1]*255)},
{int(colors(i)[2]*255)}, 1)'
    for i, cosecha in enumerate(cosechas)
}

# Iterar por cada cosecha en dataparte1
for cosecha, grupo in dataparte1.groupby('cosecha'):
    x_real = grupo['maduracion'].values
    y_real = grupo['provisionporcacum'].values
    saldo = grupo['saldo_inicial'].mean()  # saldo promedio por cosecha
    pd_val = grupo['pdponderado'].mean()   # promedio para el ajuste

    # Impacto real
    impacto_real = y_real * saldo

    # Curva ajustada
    x_range = np.linspace(0, 40, 200)
    y_pred = modelo_logistico_pd(x_range, pd_val)
    impacto_pred = y_pred * saldo

    # Color único para esta cosecha
    color = color_map[cosecha]

    # Agregar puntos reales (impacto)
    fig.add_trace(go.Scatter(x=x_real,
y=impacto_real,mode='markers+lines',name=f'{cosecha} Real',line=dict(color=color,
dash='solid'),marker=dict(size=6, color=color)))
```

```
    # Agregar curva ajustada (impacto)
    fig.add_trace(go.Scatter(x=x_range, y=impacto_pred,mode='lines',name=f'{cosecha}
Ajuste',line=dict(color=color, dash='dot')    ))

# Layout
fig.update_layout(title='Impacto Real vs Curvas Ajustadas (Provisión * Saldo
Inicial)',xaxis_title='Maduración',yaxis_title='Impacto
Monetario',template='plotly_white',legend_title='Cosecha, Tipo')
fig.show()
```



Proyección: Impacto en soles acumulado desde 2024

```
import numpy as np
import pandas as pd
import plotly.graph_objects as go

# ========================================
# 1) Filtrar data: cosechas > 202400
# ========================================
df = base[base['cosecha'] > 202400].copy()

# Asegurar tipos para evitar errores con Decimal
for col in ['provisionporcacum', 'saldo_inicial', 'pdponderado', 'maduracion']:
    df[col] = df[col].astype(float)

# ========================================
# 2) Función del modelo logístico dependiente de pdponderado
#    Usa los coeficientes del Modelo A ya obtenidos
# ========================================
def modelo_logistico_pd(x, pd):
    L = a0 + a1 * pd
    k = b0 + b1 * pd
    x0 = c0 + c1 * pd
    return L / (1 + np.exp(-k * (x - x0)))
# 3) Cálculo del REAL y PROYECTADO a nivel fila
# ========================================
# Real (preciso): porcentaje por saldo fila a fila
df['impacto_real_fila'] = df['provisionporcacum'] * df['saldo_inicial']

# Proyectado a nivel fila usando el modelo
df['porc_proyectado'] = modelo_logistico_pd(df['maduracion'], df['pdponderado'])
df['impacto_proyectado_fila'] = df['porc_proyectado'] * df['saldo_inicial']

# ========================================
# 4) Agregación por codmes
# ========================================
agg = (
    df.groupby('codmes', as_index=False)
      .agg({
          'impacto_real_fila': 'sum',
          'impacto_proyectado_fila': 'sum',
          'provisionporcacum': 'sum',
          'saldo_inicial': 'sum'
      })
)
```

```
# Método aproximado (por si lo necesitas para validar):
agg['impacto_real_aprox'] = agg['provisionporcacum'] * agg['saldo_inicial']

# Ordenar codmes como categoría ordenada para X limpio
agg['codmes'] = agg['codmes'].astype(str)
agg = agg.sort_values('codmes')

# Acumulados
agg['real_acumulado'] = agg['impacto_real_fila'].cumsum()
agg['proyectado_acumulado'] = agg['impacto_proyectado_fila'].cumsum()

# ========================================
# 5) Gráfica Plotly: Real y Proyectado
# ========================================
fig = go.Figure()

# Barras: Real mensual (preciso)
fig.add_trace(go.Bar(
    x=agg['codmes'],
    y=agg['impacto_real_fila'],
    name='Real mensual',
    marker_color='#1f77b4',  # azul suave (recuerdo tu preferencia)
    hovertemplate='Mes: %{x}<br>Real mensual: S/. %{y:,.2f}'
))

# Barras: Proyectado mensual
fig.add_trace(go.Bar(
    x=agg['codmes'],
    y=agg['impacto_proyectado_fila'],
    name='Proyectado mensual',
    marker_color='#17a2b8',  # turquesa suave
    hovertemplate='Mes: %{x}<br>Proyectado mensual: S/. %{y:,.2f}'
))

# Línea: Real acumulado
fig.add_trace(go.Scatter(
    x=agg['codmes'],
    y=agg['real_acumulado'],
    mode='lines+markers',
    name='Real acumulado',
    line=dict(color='#4e79a7', width=3, dash='dot'),
    hovertemplate='Mes: %{x}<br>Real acumulado: S/. %{y:,.2f}'
))

# Línea: Proyectado acumulado
fig.add_trace(go.Scatter(
    x=agg['codmes'],
    y=agg['proyectado_acumulado'],
    mode='lines+markers',
    name='Proyectado acumulado',
    line=dict(color='#f28e2b', width=3),  # naranja suave, pero si prefieres, lo
cambio a verde suave
    hovertemplate='Mes: %{x}<br>Proyectado acumulado: S/. %{y:,.2f}'
))

fig.update_layout(
    title='Provisión Mensual y Acumulada (Cosechas > 202400)',
    xaxis_title='Mes (codmes)',
    yaxis_title='Monto en Soles',
    template='plotly_white',
    barmode='group',          # barras lado a lado
    legend_title='Serie',
    xaxis=dict(categoryorder='array', categoryarray=list(agg['codmes']))  # asegurar
orden
)

fig.show()

# ========================================
# 6) (Opcional) Tabla resumen para exportar/validar
# ========================================
resumen = agg[['codmes', 'impacto_real_fila', 'impacto_proyectado_fila',
'real_acumulado', 'proyectado_acumulado', 'impacto_real_aprox']]
resumen.columns = ['codmes', 'real_mensual', 'proyectado_mensual', 'real_acumulado',
'proyectado_acumulado', 'real_mensual_aprox']
resumen
```

codmes	real_mensual	proyectado_mensual	real_acumulado	proyectado_acumulado	real_mensual_aprox
202401	4,858,082	3,521,120	4,858,082	3,521,120	4,858,082
202402	9,902,784	8,188,969	14,760,870	11,710,090	19,706,410
202403	14,739,020	13,412,200	29,499,880	25,122,290	44,198,800
202404	21,003,590	20,052,620	50,503,470	45,174,900	84,325,490
202405	29,087,420	27,958,460	79,590,890	73,133,360	146,958,900
202406	39,111,080	36,862,640	118,702,000	109,996,000	239,044,900
202407	49,989,360	46,640,850	168,691,300	156,636,900	355,915,000
202408	64,781,760	58,002,620	233,473,100	214,639,500	526,433,100
202409	77,591,320	70,989,100	311,064,400	285,628,600	711,659,300
202410	94,591,930	85,288,700	405,656,300	370,917,300	968,119,600
202411	108,511,600	100,554,700	514,167,900	471,472,000	1,226,836,000
202412	123,848,000	116,735,100	638,015,900	588,207,100	1,530,561,000
202501	142,829,700	134,231,100	780,845,600	722,438,200	1,926,388,000
202502	156,625,400	152,496,600	937,471,000	874,934,800	2,285,805,000
202503	175,272,500	171,503,100	1,112,743,000	1,046,438,000	2,748,735,000
202504	194,064,500	190,837,400	1,306,808,000	1,237,275,000	3,241,461,000
202505	213,108,400	210,472,900	1,519,916,000	1,447,748,000	3,779,498,000
202506	234,822,100	231,323,000	1,754,739,000	1,679,071,000	4,437,856,000
202507	253,962,100	252,077,300	2,008,701,000	1,931,148,000	5,075,134,000
202508	279,428,800	273,231,200	2,288,129,000	2,204,380,000	5,876,550,000
202509	300,058,800	295,010,600	2,588,188,000	2,499,390,000	6,653,723,000
202510	323,959,300	316,937,100	2,912,147,000	2,816,327,000	7,531,903,000