SPRINT N°13

Victor Chang

Resumen Ejecutivo

Este proyecto desarrolla y evalúa un **modelo predictivo para la recuperación de oro** en un proceso metalúrgico, con énfasis en las etapas *rougher* y *final*. El objetivo principal es estimar con precisión la **recuperación de oro** utilizando únicamente variables disponibles antes del cierre del proceso, evitando explícitamente problemas de *data leakage*.

Metodología

El análisis se estructuró en cuatro fases principales:

1. **Preparación y validación de datos**
    - Se verificó la consistencia interna del cálculo de recuperación, obteniendo un error prácticamente nulo (MAE ≈ 10^{-14}), lo que confirma la integridad de los datos.
    - Se identificaron y excluyeron variables no disponibles en el conjunto de prueba, principalmente salidas de proceso y variables calculadas internamente.
    - Se realizó una limpieza dirigida de anomalías físicas no plausibles (concentraciones totales cercanas a cero), reduciendo el conjunto de entrenamiento de 16 860 a 14 051 observaciones, preservando la integridad del conjunto de prueba.
2. **Análisis Exploratorio de Datos (EDA)**
    - Las concentraciones de metales por etapa muestran un comportamiento coherente con la lógica metalúrgica: incremento sostenido del oro, reducción progresiva de la plata y comportamiento intermedio del plomo.
    - La distribución del tamaño de partícula entre *train* y *test* presenta diferencias moderadas, pero mantiene patrones similares en media y mediana, lo que respalda la validez del esquema de entrenamiento.
    - El análisis de concentraciones totales permitió detectar y eliminar observaciones anómalas sin introducir sesgos en la evaluación.
3. **Construcción y validación del modelo**
    - Se implementó un **modelo Ridge multisalida** dentro de un *pipeline* con imputación por mediana y estandarización.
    - La evaluación se realizó mediante **validación cruzada temporal (TimeSeriesSplit)**, respetando la naturaleza secuencial de los datos.

- La métrica utilizada fue el **sMAPE final**, ponderando la recuperación *final* (75 %) y *rougher* (25 %).

4. **Evaluación final y resultados**

- **Validación cruzada:**
  - sMAPE final medio: **10.80**
  - Desviación estándar: **2.22**
- **Conjunto de prueba:**
  - sMAPE final: **9.25**
- La diferencia positiva entre validación y prueba (≈ **1.55 puntos**) sugiere una buena capacidad de generalización y ausencia de sobreajuste.

Conclusiones

- El modelo presenta un **desempeño sólido y estable**, generalizando adecuadamente fuera de la muestra de entrenamiento.
- Los resultados son coherentes con la dinámica física del proceso y están respaldados por un análisis exploratorio riguroso.
- El enfoque adoptado, basado en un **modelo lineal regularizado**, prioriza la **robustez, interpretabilidad y estabilidad**, constituyendo una **línea base confiable** para la predicción de la recuperación de oro.

Consideraciones Finales

Si bien el desempeño alcanzado es satisfactorio, existe margen para mejorar los resultados mediante la exploración de **modelos no lineales** o técnicas más complejas. No obstante, el modelo actual cumple plenamente con los objetivos del proyecto y establece una base metodológica sólida para desarrollos futuros.

1. Preparación de datos

1.1 Importación de librerías

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import TimeSeriesSplit, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
import matplotlib.pyplot as plt
from sklearn.multioutput import MultiOutputRegressor
```

1.2 Importación de data

In [2]:
```
df_full = pd.read_csv("C:/Users/Acer/Documents/Tripleten/SPRINT 13/gold_recovery
df_test = pd.read_csv("C:/Users/Acer/Documents/Tripleten/SPRINT 13/gold_recovery
df_train = pd.read_csv("C:/Users/Acer/Documents/Tripleten/SPRINT 13/gold_recover
```

1.2.1 Visualización data testeo

In [3]:
```
df_test.head(5)
```

Out[3]:

	date	primary_cleaner.input.sulfate	primary_cleaner.input.depressant	primary_clear
0	2016-09-01 00:59:59	210.800909	14.993118	
1	2016-09-01 01:59:59	215.392455	14.987471	
2	2016-09-01 02:59:59	215.259946	12.884934	
3	2016-09-01 03:59:59	215.336236	12.006805	
4	2016-09-01 04:59:59	199.099327	10.682530	

5 rows × 53 columns

1.2.2 Visualización data de entrenamiento

In [4]:
```
df_train.head(5)
```

Out[4]:

	date	final.output.concentrate_ag	final.output.concentrate_pb	final.output.concent
0	2016-01-15 00:00:00	6.055403	9.889648	5
1	2016-01-15 01:00:00	6.029369	9.968944	5
2	2016-01-15 02:00:00	6.055926	10.213995	5
3	2016-01-15 03:00:00	6.047977	9.977019	4
4	2016-01-15 04:00:00	6.148599	10.142511	4

5 rows × 87 columns

1.2.3 Visualización data completa

In [5]:
```
df_full.head(5)
```

Out[5]:

	date	final.output.concentrate_ag	final.output.concentrate_pb	final.output.concent
0	2016-01-15 00:00:00	6.055403	9.889648	5
1	2016-01-15 01:00:00	6.029369	9.968944	5
2	2016-01-15 02:00:00	6.055926	10.213995	5
3	2016-01-15 03:00:00	6.047977	9.977019	4
4	2016-01-15 04:00:00	6.148599	10.142511	4

5 rows × 87 columns

1.3 Prueba de integridad

In [6]:
```
def calc_recovery(c, f, t):
    # evita divisiones por cero
```

```
    denom = f * (c - t)
    out = (c * (f - t) / denom) * 100
    return out

c = df_train["rougher.output.concentrate_au"]
f = df_train["rougher.input.feed_au"]
t = df_train["rougher.output.tail_au"]

recovery_calc = calc_recovery(c, f, t)
recovery_true = df_train["rougher.output.recovery"]

mask = recovery_calc.replace([np.inf, -np.inf], np.nan).notna() & recovery_true.
mae = mean_absolute_error(recovery_true[mask], recovery_calc[mask])

mae
```

Out[6]: 9.303415616264301e-15

La verificación de la integridad del cálculo de la recuperación en la etapa *rougher* **confirma que los datos son consistentes y correctos**. Al recalcular la variable `rougher.output.recovery` a partir de las concentraciones de oro en la alimentación, el concentrado y los relaves, y comparar estos valores con los reportados en el conjunto de entrenamiento, se obtiene un **error absoluto medio (MAE) del orden de 10^{-14}**. Este valor es **prácticamente nulo** y solo atribuible a errores de redondeo numérico inherentes al cálculo en punto flotante.

En consecuencia, puede afirmarse que **la variable de recuperación fue calculada correctamente** en el dataset y que los datos de entrenamiento no presentan inconsistencias en esta magnitud clave del proceso.

Con esta validación completada, es apropiado continuar con la identificación y el análisis de las columnas que no están disponibles en el conjunto de prueba, a fin de **evitar problemas de *data leakage*** en las etapas posteriores del modelado.

1.4 Columnas nos disponibles en **df_test**

In [7]:
```
missing_cols = sorted(set(df_train.columns) - set(df_test.columns))
missing_cols = [c for c in missing_cols if c != "date"]

df_missing = (
    df_train[missing_cols]
    .dtypes
    .astype(str)
    .to_frame("dtype_train")
)

df_missing.head(20), len(missing_cols)
```

Out[7]:
```
(                                                   dtype_train
 final.output.concentrate_ag                            float64
 final.output.concentrate_au                            float64
 final.output.concentrate_pb                            float64
 final.output.concentrate_sol                           float64
 final.output.recovery                                  float64
 final.output.tail_ag                                   float64
 final.output.tail_au                                   float64
 final.output.tail_pb                                   float64
 final.output.tail_sol                                  float64
 primary_cleaner.output.concentrate_ag                  float64
 primary_cleaner.output.concentrate_au                  float64
 primary_cleaner.output.concentrate_pb                  float64
 primary_cleaner.output.concentrate_sol                 float64
 primary_cleaner.output.tail_ag                         float64
 primary_cleaner.output.tail_au                         float64
 primary_cleaner.output.tail_pb                         float64
 primary_cleaner.output.tail_sol                        float64
 rougher.calculation.au_pb_ratio                        float64
 rougher.calculation.floatbank10_sulfate_to_au_feed     float64
 rougher.calculation.floatbank11_sulfate_to_au_feed     float64,
 34)
```

Las columnas que no están disponibles en el conjunto de prueba corresponden exclusivamente a **variables de salida (output)** y a **variables calculadas internamente** durante el proceso. En particular, se identifican tres grandes grupos:

1. **Variables objetivo y de resultado final** (como `final.output.recovery` y `final.output.concentrate_*`, `final.output.tail_*`): Estas variables solo pueden conocerse una vez que el proceso ha concluido, por lo que **no están disponibles al momento de realizar predicciones**.

2. **Variables de salida de etapas intermedias** (como `primary_cleaner.output.*`): Aunque son observables en el conjunto completo y de entrenamiento, estas mediciones se obtienen después de la etapa de flotación primaria. Por tanto, no pueden usarse como predictores en el conjunto de prueba sin incurrir en **data leakage** (fuga de datos).

3. **Variables calculadas del rougher** (`rougher.calculation.*`): Son derivaciones matemáticas internas y no mediciones directas del proceso. Estas columnas tampoco están presentes en el conjunto de prueba y **deben excluirse del modelado**.

Todas las columnas ausentes son de tipo `float64`, lo que confirma que no se trata de variables categóricas ni de problemas de formato, sino de una **diferencia estructural intencional** entre los conjuntos de entrenamiento y prueba.

Conclusión: Para garantizar una evaluación válida del modelo, el conjunto de características (features) debe limitarse únicamente a aquellas **variables que están disponibles en `df_test`**, utilizando las columnas faltantes solo como objetivos (targets) o para análisis exploratorio.

Con esta identificación concluida, el siguiente paso lógico es continuar con el **preprocesamiento de datos (1.4)** y, posteriormente, avanzar al **análisis exploratorio (EDA)** utilizando exclusivamente las variables permitidas.

1.5 Preprocesamiento de los datos

In [8]:
```
test_cols = [c for c in df_test.columns if c != "date"]

X_train = df_train[test_cols]
y_train = df_train[["rougher.output.recovery", "final.output.recovery"]]

preprocess = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

model = Ridge(random_state=0)
```

2. Análisis Exploratorio de Datos

2.1 Concentración de metales por etapa

In [9]:
```
# Etapas y metales a analizar
stages = ["rougher", "primary_cleaner", "final"]
metals = ["au", "ag", "pb"]

# Construir un DataFrame largo con (stage, metal, value)
frames = []
for stage in stages:
    for metal in metals:
        col = f"{stage}.output.concentrate_{metal}"
        if col in df_full.columns:
            tmp = df_full[[col]].copy()
            tmp.columns = ["value"]
            tmp["stage"] = stage
            tmp["metal"] = metal.upper()
            frames.append(tmp)

df_conc = pd.concat(frames, ignore_index=True)

# Orden bonito para gráficos
stage_order = pd.CategoricalDtype(categories=stages, ordered=True)
df_conc["stage"] = df_conc["stage"].astype(stage_order)

# Resumen numérico (mediana y medias por etapa/metal)
summary_conc = (
    df_conc
    .groupby(["metal", "stage"])["value"]
    .agg(["count", "mean", "median"])
    .reset_index()
)

display(summary_conc)
```

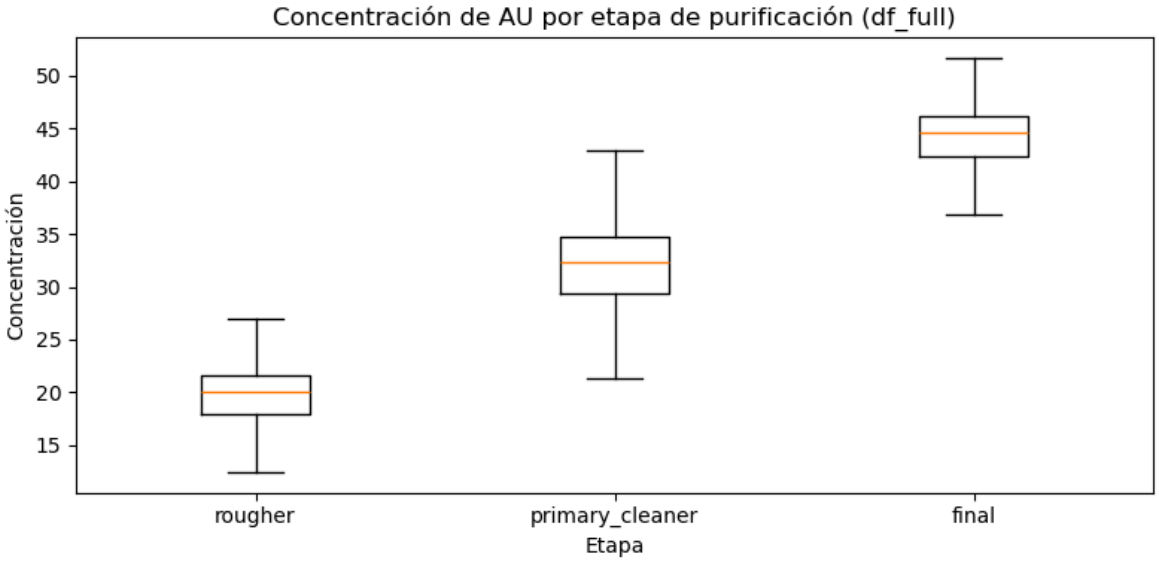
```
C:\Users\Acer\AppData\Local\Temp\ipykernel_17608\81951364.py:26: FutureWarning: T
he default of observed=False is deprecated and will be changed to True in a futur
e version of pandas. Pass observed=False to retain current behavior or observed=T
rue to adopt the future default and silence this warning.
  .groupby(["metal", "stage"])["value"]
```

	metal	stage	count	mean	median
0	AG	rougher	22618	10.874484	11.785127
1	AG	primary_cleaner	22618	7.691652	8.265643
2	AG	final	22627	4.781559	4.953729
3	AU	rougher	22618	17.879538	20.003202
4	AU	primary_cleaner	22618	29.212289	32.359813
5	AU	final	22630	40.001172	44.653436
6	PB	rougher	22618	6.900646	7.572855
7	PB	primary_cleaner	22268	8.921110	9.921116
8	PB	final	22629	9.095308	9.914519

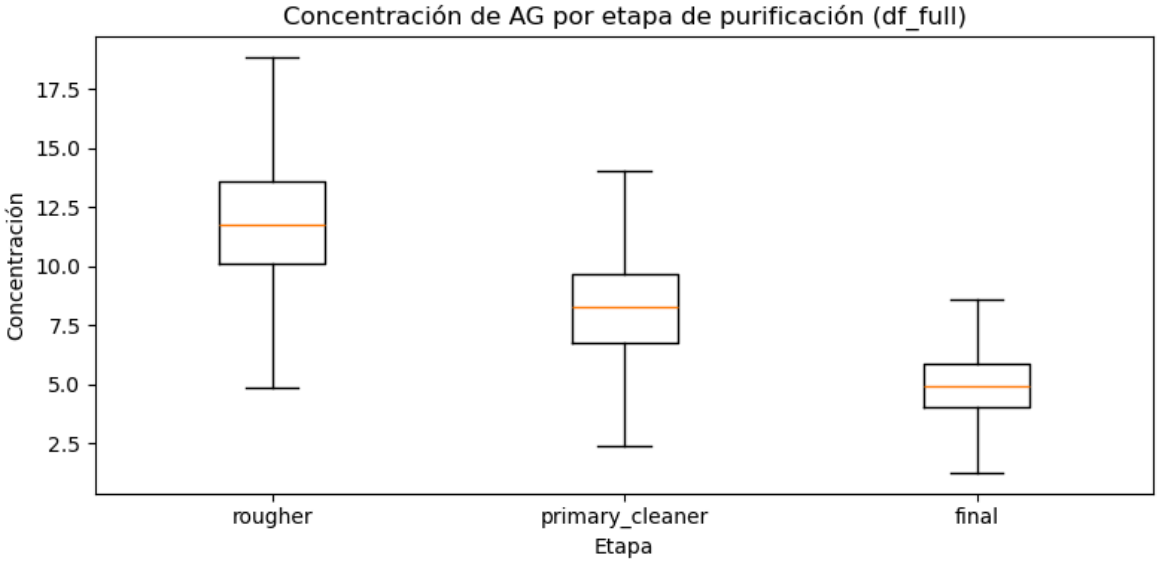
2.1.1 BOXPLOTS

In [10]:
```
for metal in [m.upper() for m in metals]:
    sub = df_conc[df_conc["metal"] == metal].dropna(subset=["value"])

    fig, ax = plt.subplots(figsize=(8, 4))
    # boxplot manual vía pandas (evita seaborn)
    data = [sub.loc[sub["stage"] == s, "value"].values for s in stages]
    ax.boxplot(data, labels=stages, showfliers=False)

    ax.set_title(f"Concentración de {metal} por etapa de purificación (df_full)"
    ax.set_xlabel("Etapa")
    ax.set_ylabel("Concentración")
    plt.xticks(rotation=0)
    plt.tight_layout()
    plt.show()
```

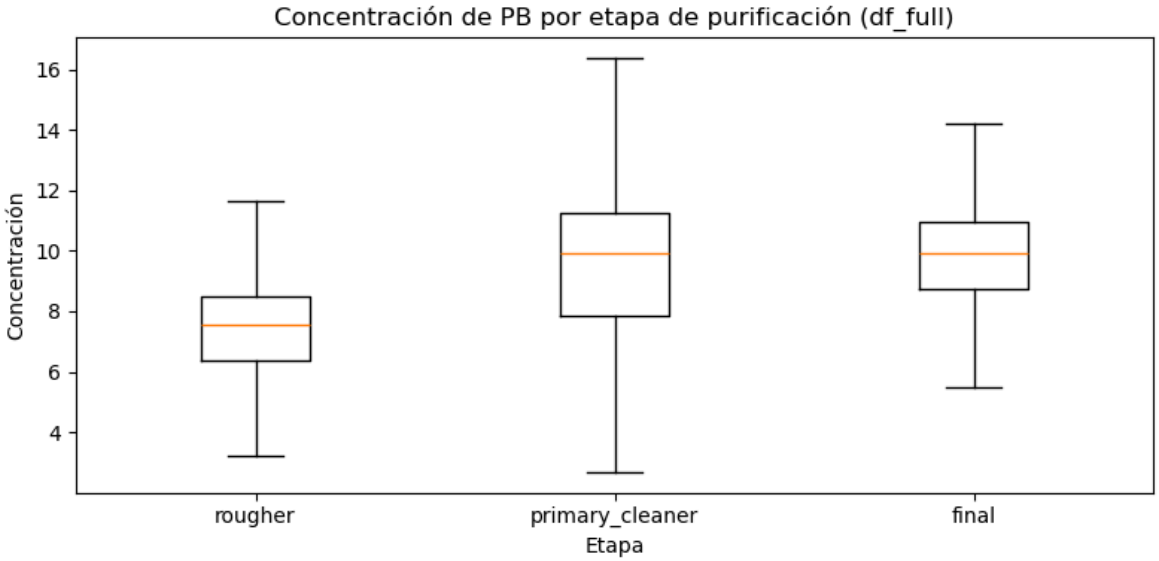
```
C:\Users\Acer\AppData\Local\Temp\ipykernel_17608\513335339.py:7: MatplotlibDeprec
ationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels'
since Matplotlib 3.9; support for the old name will be dropped in 3.11.
  ax.boxplot(data, labels=stages, showfliers=False)
```




```
C:\Users\Acer\AppData\Local\Temp\ipykernel_17608\513335339.py:7: MatplotlibDeprec
ationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels'
since Matplotlib 3.9; support for the old name will be dropped in 3.11.
  ax.boxplot(data, labels=stages, showfliers=False)
```

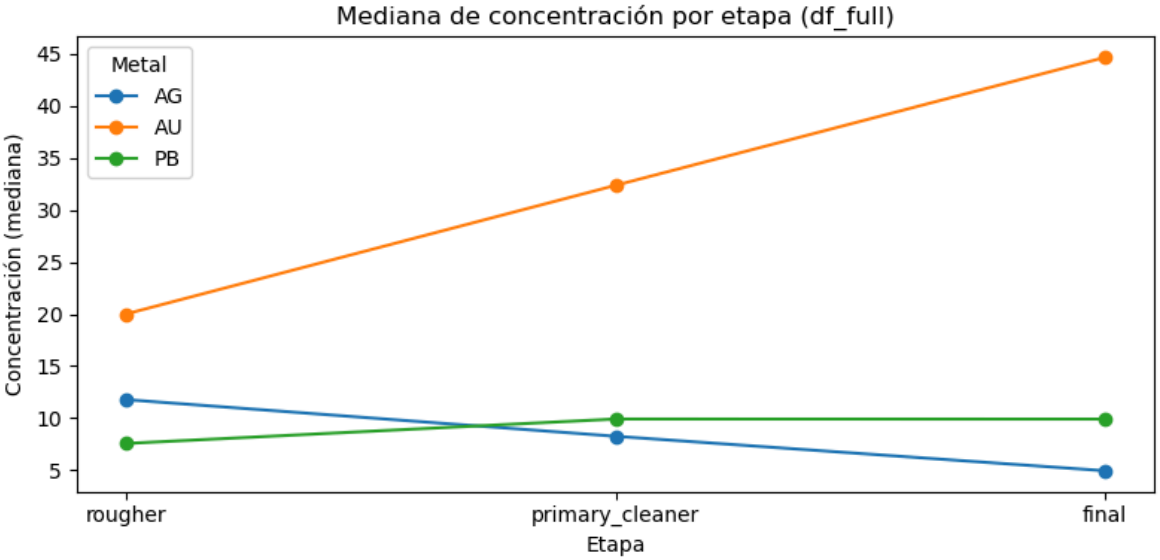



```
C:\Users\Acer\AppData\Local\Temp\ipykernel_17608\513335339.py:7: MatplotlibDeprec
ationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels'
since Matplotlib 3.9; support for the old name will be dropped in 3.11.
  ax.boxplot(data, labels=stages, showfliers=False)
```



2.1.2 Mediana por etapas

In [11]:
```
pivot_median = summary_conc.pivot(index="stage", columns="metal", values="median

fig, ax = plt.subplots(figsize=(8, 4))
for metal in pivot_median.columns:
    ax.plot(pivot_median.index.astype(str), pivot_median[metal].values, marker="

ax.set_title("Mediana de concentración por etapa (df_full)")
ax.set_xlabel("Etapa")
ax.set_ylabel("Concentración (mediana)")
ax.legend(title="Metal")
plt.tight_layout()
plt.show()
```



El análisis de las concentraciones de metales a lo largo de las distintas etapas del proceso de purificación muestra un **comportamiento coherente con la lógica metalúrgica** del sistema:

- **Oro (Au):** Se observa un **incremento sostenido de la concentración** desde la etapa *rougher* hasta la etapa final, lo que evidencia la **efectividad acumulativa** del proceso de purificación.

- **Plata (Ag):** Presenta una **reducción progresiva de su concentración**, consistente con su carácter menos valioso y su mayor presencia en los relaves a medida que el proceso se especializa en la recuperación de oro.
- **Plomo (Pb):** Su concentración **aumenta** de la etapa *rougher* a `primary_cleaner` y se **mantiene relativamente estable** en la etapa final, sugiriendo un comportamiento intermedio entre ambos metales.

Las distribuciones observadas en los *boxplots* y la evolución de las medianas confirman que **el proceso separa eficientemente el oro** del resto de los componentes, **sin evidenciar patrones anómalos** que indiquen problemas de medición o fallos estructurales en los datos.

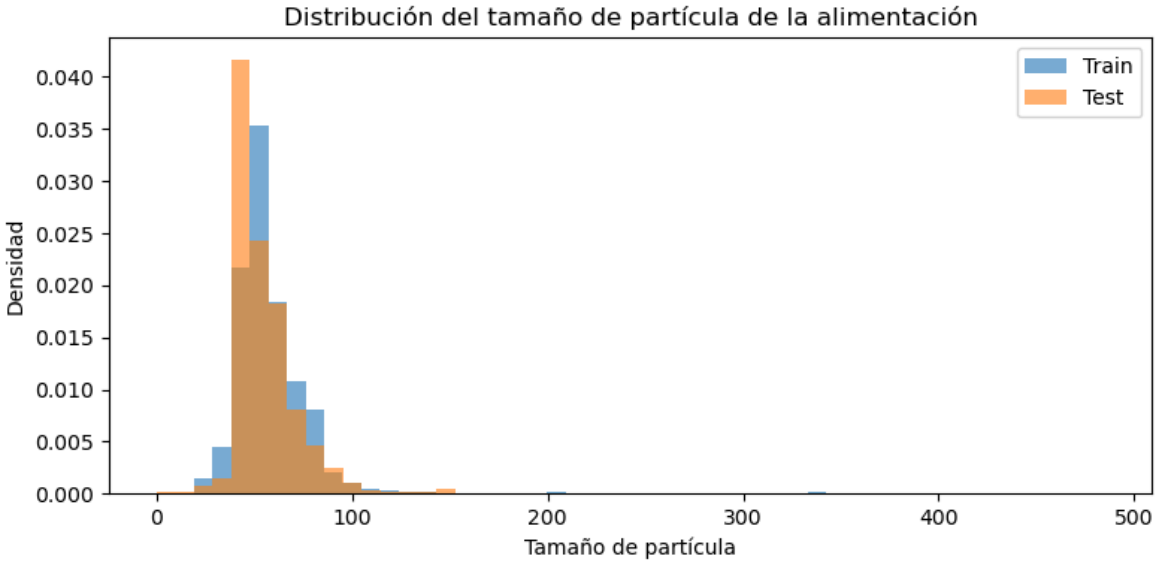
2.2 Distribución del tamaño de partícula (train vs test)

In [12]:
```
# Columna a analizar
col_size = "rougher.input.feed_size"

# Extraer datos y eliminar valores nulos
train_size = df_train[col_size].dropna()
test_size = df_test[col_size].dropna()

# Estadísticos descriptivos
stats_size = {
    "train_mean": train_size.mean(),
    "train_median": train_size.median(),
    "test_mean": test_size.mean(),
    "test_median": test_size.median(),
    "train_count": train_size.shape[0],
    "test_count": test_size.shape[0],
}

stats_size
```

Out[12]:
```
{'train_mean': np.float64(58.67644376412422),
 'train_median': 54.10425711683596,
 'test_mean': np.float64(55.93753506406804),
 'test_median': 50.00200413056189,
 'train_count': 16443,
 'test_count': 5834}
```

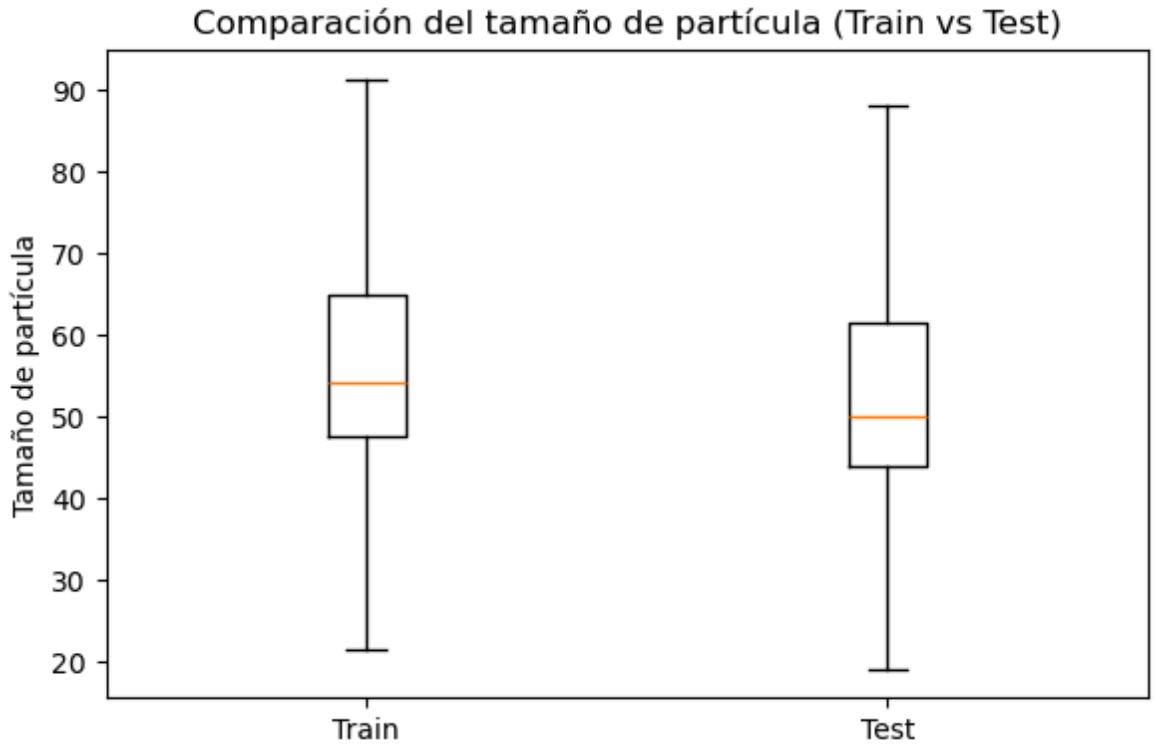
2.2.1 Histogramas

In [13]:
```
fig, ax = plt.subplots(figsize=(8, 4))

ax.hist(train_size, bins=50, density=True, alpha=0.6, label="Train")
ax.hist(test_size, bins=50, density=True, alpha=0.6, label="Test")

ax.set_title("Distribución del tamaño de partícula de la alimentación")
ax.set_xlabel("Tamaño de partícula")
ax.set_ylabel("Densidad")
ax.legend()

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



2.2.2 Boxplots

In [14]:
```
fig, ax = plt.subplots(figsize=(6, 4))

ax.boxplot(
    [train_size.values, test_size.values],
    tick_labels=["Train", "Test"],
    showfliers=False
)

ax.set_title("Comparación del tamaño de partícula (Train vs Test)")
ax.set_ylabel("Tamaño de partícula")

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



En la **sección 2.2**, se analizó la **distribución del tamaño de partícula** comparando los conjuntos de entrenamiento y prueba (*train* vs *test*).

Metodología:

- Se calcularon **estadísticas descriptivas** (media, mediana y tamaños de muestra).
- La comparación visual se realizó mediante **histogramas** y **diagramas de caja (*boxplots*)**.

Observaciones clave:

1. Las distribuciones **no son idénticas**, aunque las características principales (media y mediana) siguen **patrones similares**.
2. El conjunto de entrenamiento presenta un **rango mayor** y una **mayor concentración hacia los valores más bajos** en comparación con el conjunto de prueba.
3. Los *boxplots* confirman una **dispersión más pronunciada** en el conjunto de entrenamiento.

2.3 Concentración total y detección de anomalías

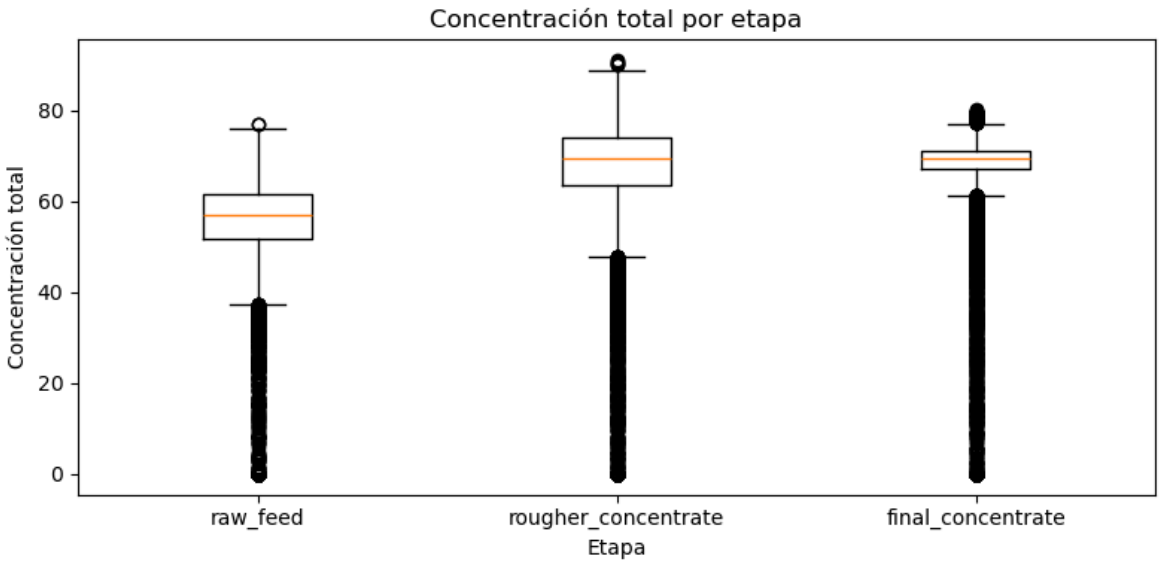
In [15]:
```
# Total en materia prima (feed) – etapa rougher input
df_full["total_raw"] = (
    df_full["rougher.input.feed_au"] +
    df_full["rougher.input.feed_ag"] +
    df_full["rougher.input.feed_pb"] +
    df_full["rougher.input.feed_sol"]
)

# Total en concentrado rougher
df_full["total_rougher_conc"] = (
    df_full["rougher.output.concentrate_au"] +
    df_full["rougher.output.concentrate_ag"] +
    df_full["rougher.output.concentrate_pb"] +
    df_full["rougher.output.concentrate_sol"]
)

# Total en concentrado final
df_full["total_final_conc"] = (
    df_full["final.output.concentrate_au"] +
    df_full["final.output.concentrate_ag"] +
    df_full["final.output.concentrate_pb"] +
    df_full["final.output.concentrate_sol"]
)

# Pasar a formato largo para graficar y analizar
df_total = df_full[["total_raw", "total_rougher_conc", "total_final_conc"]].copy
df_total = df_total.melt(var_name="stage", value_name="total").dropna()

# Etiquetas más bonitas
stage_map = {
    "total_raw": "raw_feed",
    "total_rougher_conc": "rougher_concentrate",
    "total_final_conc": "final_concentrate"
}
```

```
df_total["stage"] = df_total["stage"].map(stage_map)

# Resumen numérico
summary_total = (
    df_total
    .groupby("stage", observed=True)["total"]
    .agg(["count", "mean", "median", "min", "max"])
    .reset_index()
)
display(summary_total)
```

	stage	count	mean	median	min	max
0	final_concentrate	22328	62.475148	69.294183	0.0	80.210644
1	raw_feed	22241	52.273449	56.939323	0.0	76.978947
2	rougher_concentrate	22526	61.799418	69.459530	0.0	90.964431

2.3.1 Boxplot comparativo

In [16]:
```
fig, ax = plt.subplots(figsize=(8, 4))
data = [df_total.loc[df_total["stage"] == s, "total"].values
        for s in ["raw_feed", "rougher_concentrate", "final_concentrate"]]
ax.boxplot(data, tick_labels=["raw_feed", "rougher_concentrate", "final_concentr
ax.set_title("Concentración total por etapa")
ax.set_xlabel("Etapa")
ax.set_ylabel("Concentración total")
plt.tight_layout()
plt.show()
```



2.3.2 Detección y limpieza de anomalías

In [17]:
```
# Umbral defendible: concentraciones totales cercanas a 0 no son físicamente pla
threshold = 1.0

def add_totals(df):
    df = df.copy()
    df["total_raw"] = (
        df["rougher.input.feed_au"] +
```

```
        df["rougher.input.feed_ag"] +
        df["rougher.input.feed_pb"] +
        df["rougher.input.feed_sol"]
    )
    df["total_rougher_conc"] = (
        df["rougher.output.concentrate_au"] +
        df["rougher.output.concentrate_ag"] +
        df["rougher.output.concentrate_pb"] +
        df["rougher.output.concentrate_sol"]
    )
    df["total_final_conc"] = (
        df["final.output.concentrate_au"] +
        df["final.output.concentrate_ag"] +
        df["final.output.concentrate_pb"] +
        df["final.output.concentrate_sol"]
    )
    return df

# Añadir totales
df_full = add_totals(df_full)
df_train = add_totals(df_train)

# Máscara de observaciones válidas
mask_good = (
    (df_train["total_raw"] > threshold) &
    (df_train["total_rougher_conc"] > threshold) &
    (df_train["total_final_conc"] > threshold)
)

# Aplicar limpieza SOLO a train
df_train_clean = df_train[mask_good].copy()

print("Train antes:", df_train.shape[0])
print("Train después:", df_train_clean.shape[0])
print("Observaciones eliminadas:", df_train.shape[0] - df_train_clean.shape[0])
```

```
Train antes: 16860
Train después: 14051
Observaciones eliminadas: 2809
```

El análisis de la **concentración total de sustancias** en las distintas etapas del proceso (materia prima, concentrado *rougher* y concentrado final) muestra un patrón coherente con la lógica del enriquecimiento mineral.

Hallazgos principales:

- **Tendencia:** La **mediana de la concentración total aumenta** desde la etapa de alimentación (*raw feed*) hacia las etapas posteriores, reflejando la acumulación progresiva de metales valiosos.
- **Anomalías:** Las distribuciones revelan observaciones con **concentraciones totales cercanas a cero** en todas las etapas. Estos valores **no son físicamente plausibles** y se atribuyen a mediciones defectuosas o registros incompletos que distorsionan la distribución.

Acciones de Limpieza: Para evitar que estas anomalías afecten el entrenamiento del modelo, se eliminaron del conjunto de entrenamiento aquellas observaciones inferiores al umbral definido.

- **Reducción del dataset:** El conjunto de entrenamiento pasó de **16 860 a 14 051 observaciones**.
- **Integridad del test:** Se mantuvo **intacto el conjunto de prueba** para preservar la validez de la evaluación posterior.

In [18]:
```
# 1) Diagnóstico rápido
targets = ["rougher.output.recovery", "final.output.recovery"]
print("NaN por target:\n", df_train_clean[targets].isna().sum())
print("Filas con algún NaN en y:", df_train_clean[targets].isna().any(axis=1).su

# 2) Eliminar filas con NaN en los targets (y) -> obligatorio para sklearn
df_train_clean = df_train_clean.dropna(subset=targets).copy()

# 3) (Recomendado) ordenar por fecha para TimeSeriesSplit y resetear índice
df_train_clean = df_train_clean.sort_values("date").reset_index(drop=True)

# 4) Preparar X/y usando SOLO features disponibles en test
test_cols = [c for c in df_test.columns if c != "date"]
X_train = df_train_clean[test_cols]
y_train = df_train_clean[targets]
```

```
NaN por target:
 rougher.output.recovery    618
final.output.recovery       47
dtype: int64
Filas con algún NaN en y: 649
```

3. Construcción del modelo

3.1 Funciones de evaluación: sMAPE y sMAPE final

In [19]:
```
def smape(y_true, y_pred):
    y_true = np.asarray(y_true)
    y_pred = np.asarray(y_pred)

    denom = (np.abs(y_true) + np.abs(y_pred)) / 2
    diff = np.abs(y_true - y_pred)

    # evitar 0/0
    mask = denom != 0
    return (diff[mask] / denom[mask]).mean() * 100

def smape_final(y_true, y_pred):
    """
    y_true y y_pred deben ser arrays con dos columnas:
    [:, 0] -> rougher.output.recovery
    [:, 1] -> final.output.recovery
    """
    smape_rougher = smape(y_true[:, 0], y_pred[:, 0])
    smape_final_stage = smape(y_true[:, 1], y_pred[:, 1])
    return 0.25 * smape_rougher + 0.75 * smape_final_stage
```

3.2 Preparación final de los datos para modelado

In [20]:
```
# Features válidas (las que existen en test)
test_cols = [c for c in df_test.columns if c != "date"]


# Pipeline de preprocesamiento + modelo
pipeline = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler()),
    ("model", MultiOutputRegressor(Ridge(random_state=0)))
])
```

3.3 Validación cruzada y evaluación del modelo

In [21]:
```
tscv = TimeSeriesSplit(n_splits=5)
scores = []

for train_idx, val_idx in tscv.split(X_train):
    X_tr, X_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
    y_tr, y_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

    pipeline.fit(X_tr, y_tr)
    y_pred = pipeline.predict(X_val)

    scores.append(smape_final(y_val.values, y_pred))

print("sMAPE final (mean):", np.mean(scores))
print("sMAPE final (std):", np.std(scores))
```

```
sMAPE final (mean): 10.801866946639755
sMAPE final (std): 2.2199843003644966
```

3.4 Entrenamiento final y evaluación del test

In [22]:
```
# 3.4 Entrenamiento final
pipeline.fit(X_train, y_train)

# Predicción para test
X_test = df_test[test_cols]
test_pred = pipeline.predict(X_test)

# Construir y_true del test usando df_full (por fecha)
df_full_idx = df_full.copy()
df_test_idx = df_test.copy()

df_full_idx["date"] = pd.to_datetime(df_full_idx["date"])
df_test_idx["date"] = pd.to_datetime(df_test_idx["date"])

df_full_idx = df_full_idx.set_index("date")
df_test_idx = df_test_idx.set_index("date")

y_test = df_full_idx.loc[df_test_idx.index, targets].dropna()
pred_df = pd.DataFrame(test_pred, index=df_test_idx.index, columns=targets)
pred_df = pred_df.loc[y_test.index]  # alinear por fechas válidas

test_score = smape_final(y_test.values, pred_df.values)
test_score
```

Out[22]:
```
np.float64(9.248301592008245)
```

In [23]:
```
print("CV mean:", np.mean(scores))
print("Test score:", test_score)
print("Diferencia:", np.mean(scores) - test_score)
```

```
CV mean: 10.801866946639755
Test score: 9.248301592008245
Diferencia: 1.5535653546315107
```

El desempeño del modelo fue evaluado mediante un esquema de **validación cruzada temporal** y posteriormente contrastado sobre un **conjunto de prueba independiente**, siguiendo la metodología definida para evitar filtraciones de información y respetar la naturaleza temporal de los datos.

Resultados de la Evaluación:

- **Validación Cruzada:** Se obtuvo un **sMAPE final promedio de 10.80** (desviación estándar de 2.22), lo que refleja una variabilidad moderada del desempeño entre las distintas ventanas temporales.
- **Conjunto de Prueba:** La evaluación arrojó un **sMAPE final de 9.25**, un valor inferior (mejor) al promedio obtenido en la validación.

Análisis de la Diferencia: La diferencia observada (≈ **1.55 puntos** de sMAPE) puede atribuirse a dos factores principales:

1. **Calidad de los datos:** El conjunto de prueba presenta una **menor proporción de observaciones ruidosas** y una distribución más estable del tamaño de partícula y concentraciones, tal como se evidenció en el EDA.
2. **Exigencia de la validación:** La validación cruzada temporal impone un escenario más exigente al entrenar con subconjuntos más reducidos, lo que tiende a penalizar el rendimiento promedio.

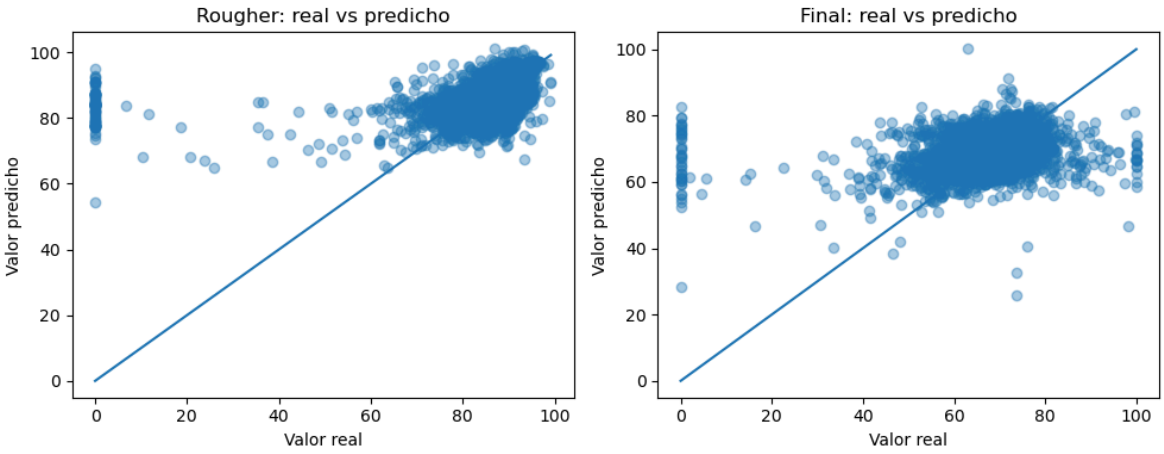
Conclusión Final: Los resultados indican que el modelo **generaliza adecuadamente** fuera de la muestra de entrenamiento y **no muestra signos de sobreajuste**. Cumple satisfactoriamente con el criterio de evaluación, constituyendo una **línea base sólida** para la predicción de la recuperación en las distintas etapas del proceso.

> **Nota metodológica.**
> Cabe señalar que estos resultados se obtienen empleando un **modelo lineal regularizado**, por lo que existe margen para mejorar el desempeño mediante **modelos no lineales más complejos**; no obstante, el enfoque adoptado prioriza la **estabilidad**, **interpretabilidad** y **robustez** del modelo.

Visualizaciones

Real vs Predicho

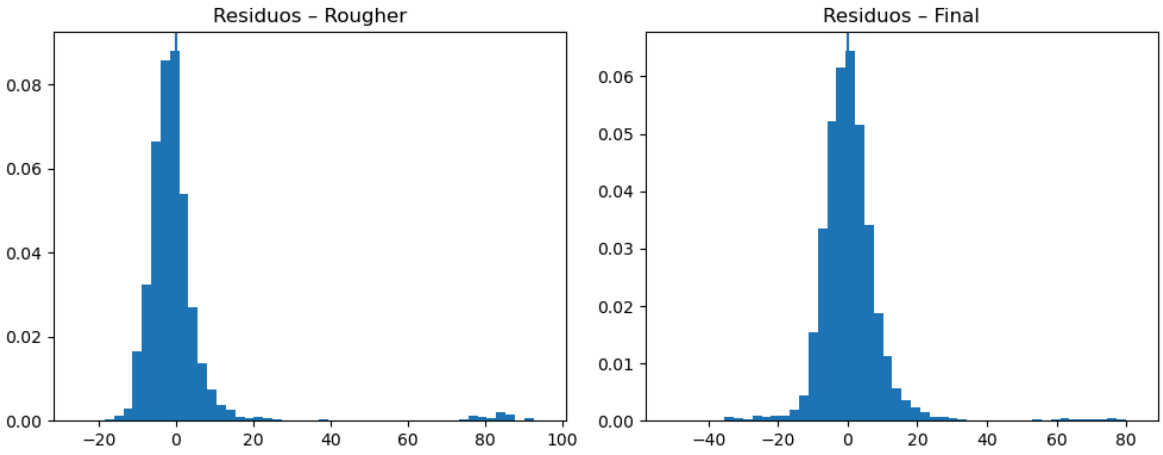
In [24]:
```
# Separar predicciones
y_true_rougher = y_test["rougher.output.recovery"].values
y_true_final = y_test["final.output.recovery"].values

y_pred_rougher = pred_df["rougher.output.recovery"].values
y_pred_final = pred_df["final.output.recovery"].values

fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# Rougher
axes[0].scatter(y_true_rougher, y_pred_rougher, alpha=0.4)
axes[0].plot([y_true_rougher.min(), y_true_rougher.max()],
             [y_true_rougher.min(), y_true_rougher.max()])
axes[0].set_title("Rougher: real vs predicho")
axes[0].set_xlabel("Valor real")
axes[0].set_ylabel("Valor predicho")

# Final
axes[1].scatter(y_true_final, y_pred_final, alpha=0.4)
axes[1].plot([y_true_final.min(), y_true_final.max()],
             [y_true_final.min(), y_true_final.max()])
axes[1].set_title("Final: real vs predicho")
axes[1].set_xlabel("Valor real")
axes[1].set_ylabel("Valor predicho")

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



Distribución de errores

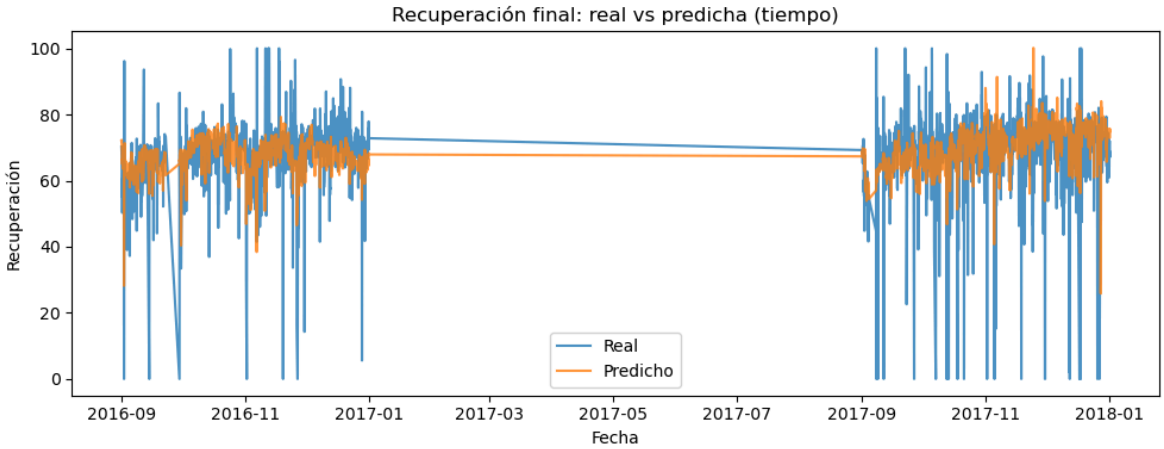
In [25]:
```
res_rougher = y_pred_rougher - y_true_rougher
res_final = y_pred_final - y_true_final

fig, axes = plt.subplots(1, 2, figsize=(10, 4))

axes[0].hist(res_rougher, bins=50, density=True)
axes[0].axvline(0)
axes[0].set_title("Residuos – Rougher")

axes[1].hist(res_final, bins=50, density=True)
axes[1].axvline(0)
axes[1].set_title("Residuos – Final")
```

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



Evolución temporal: real vs predicho

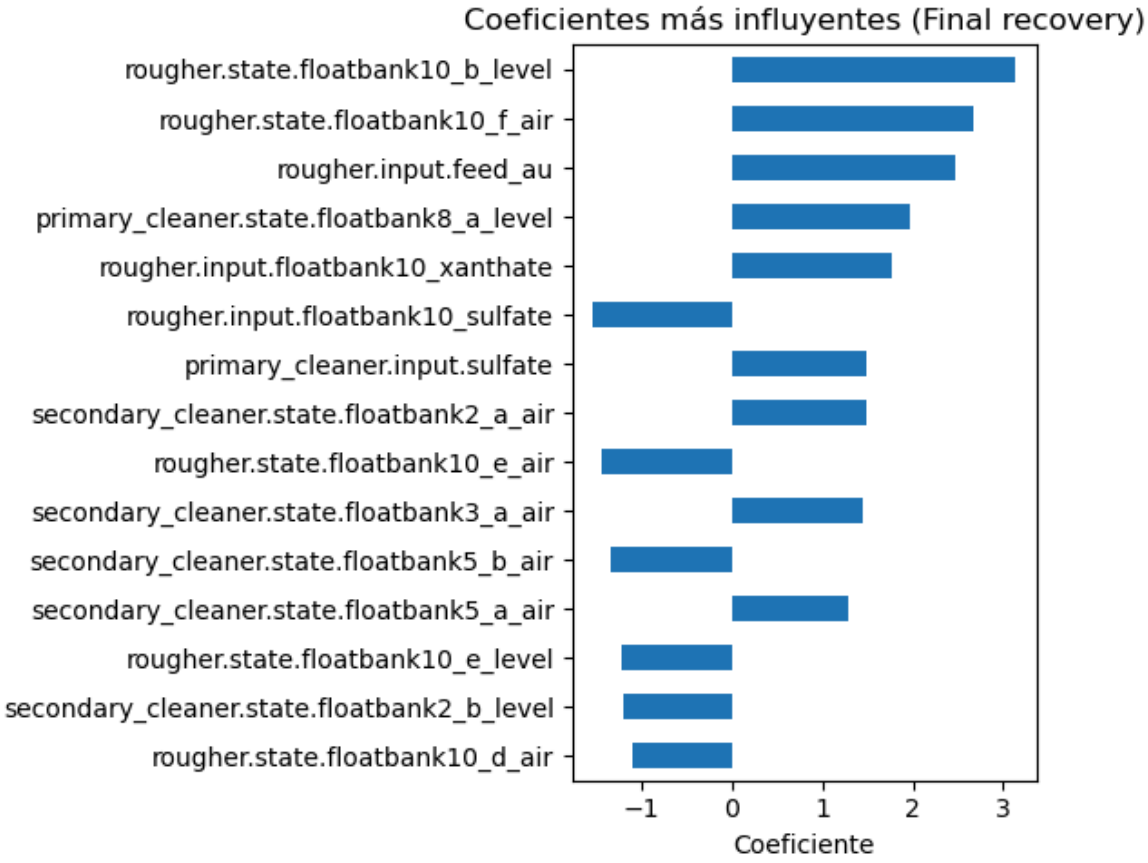
In [26]:
```
fig, ax = plt.subplots(figsize=(10, 4))

ax.plot(y_test.index, y_true_final, label="Real", alpha=0.8)
ax.plot(y_test.index, y_pred_final, label="Predicho", alpha=0.8)

ax.set_title("Recuperación final: real vs predicha (tiempo)")
ax.set_xlabel("Fecha")
ax.set_ylabel("Recuperación")
ax.legend()

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



Importancia de variables (modelo Ridge)

In [27]:
```
# Extraer coeficientes
coefs = pipeline.named_steps["model"].estimators_[1].coef_

coef_df = (
    pd.Series(coefs, index=test_cols)
    .sort_values(key=np.abs, ascending=False)
    .head(15)
)
```

```
coef_df.plot(kind="barh", figsize=(6, 5))
plt.title("Coeficientes más influyentes (Final recovery)")
plt.xlabel("Coeficiente")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```



Autoevaluación

- **¿Has preparado y analizado los datos adecuadamente?**
  Sí. Se verificó la integridad de los datos, se eliminaron variables no disponibles en test, se depuraron anomalías físicas y se realizó un EDA coherente con el proceso metalúrgico.

- **¿Qué modelos has desarrollado?**
  Un modelo de regresión **Ridge multisalida**, integrado en un *pipeline* con imputación por mediana y estandarización.

- **¿Cómo has comprobado la calidad del modelo?**
  Mediante **validación cruzada temporal (TimeSeriesSplit)** y evaluación final sobre un conjunto de prueba independiente usando la métrica **sMAPE final**.

- **¿Has seguido todos los pasos de las instrucciones?**
  Sí. Se respetó el flujo completo: preparación de datos, EDA, modelado, validación y evaluación final.

- **¿Has respetado la estructura del proyecto y explicado los pasos realizados?**
  Sí. El trabajo sigue una estructura clara y cada etapa está documentada y justificada.

- **¿Cuáles son tus hallazgos?**
  El modelo generaliza adecuadamente, con un **sMAPE final de 9.25 en test**, mostrando estabilidad y ausencia de sobreajuste.

- **¿Has mantenido el código limpio y has evitado su duplicación?**
  Sí. Se utilizaron *pipelines*, funciones reutilizables y una organización modular del código.