

El esqueleto — cópialo tal cual

```
import numpy as np, pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import (HistGradientBoostingClassifier,
                              HistGradientBoostingRegressor)

CAT = ["contrato", "region"] # texto
NUM = ["precio", "meses", "tickets"] # números

pre = ColumnTransformer([
    ("cat", OneHotEncoder(handle_unknown="ignore",
                          sparse_output=False), CAT),
    ("num", "passthrough", NUM),
])

model = Pipeline([
    ("pre", pre),
    ("m", HistGradientBoostingClassifier(random_state=0)),
])

model.fit(X[CAT + NUM], y)
pred = model.predict_proba(X_score[CAT + NUM][:, 1])
# regresión: np.clip(model.predict(X_score[CAT+NUM]), 0, None)

Un solo fit, sobre el train; el holdout sólo se transforma. Lo que no listes en CAT/NUM no entra: ahí se quedan el id, lo constante y lo que se escribe después del desenlace.
```

VARIANTE modelos lineales, kNN, SVM

HistGB no necesita nada; el resto sí. Imputar fuera con la media de *todo* el archivo es fuga.

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

num = Pipeline([("imp", SimpleImputer(strategy="median")),
                ("sc", StandardScaler())])
cat = Pipeline([("imp", SimpleImputer(strategy="most_frequent")),
                ("oh", OneHotEncoder(handle_unknown="ignore",
                                     sparse_output=False))])
pre = ColumnTransformer([("num", num, NUM), ("cat", cat, CAT)])
```

LAS CINCO FORMAS DEL PROBLEMA

SITE PIDEN	MODELO · ÚLTIMA LÍNEA · MÉTRICA · PARTICIÓN
Ordenar por riesgo (churn, fraude)	Classifier · predict_proba(X)[: ,1] · roc_auc_score · stratify=y
Un número en el futuro (demanda, precio)	Regressor · np.clip(predict(X), 0, None) · RMSE · corte por fecha
Clase rara (≈5%) + presupuesto	Classifier · predict_proba(X)[: ,1] · average_precision_score + recall@k · stratify=y
Varias clases una de ellas rara	Classifier · predict(X) · f1_score(average="macro") · stratify=y
Varias filas por persona / lote	el que toque · según · según · GroupShuffleSplit por esa entidad

LAS TRES PARTICIONES

```
# 1 · al azar, con la proporción de clases intacta
from sklearn.model_selection import train_test_split
X_tr, X_va, y_tr, y_va = train_test_split(X, y, test_size=0.25, stratify=y, random_state=0)

# 2 · por tiempo – fecha frontera, NUNCA iloc[-n:]
d = pd.to_datetime(df["fecha"], format="mixed", dayfirst=True)
b = d.max() - pd.Timedelta(days=n_dias - 1)
train, valid = df[d < b].copy(), df[d >= b].copy()

# 3 · por grupo – ninguna entidad en los dos lados
from sklearn.model_selection import GroupShuffleSplit
i_tr, i_va = next(GroupShuffleSplit(n_splits=1, test_size=0.25, random_state=0).split(X, y, groups=df["patient_id"]))
train, valid = df.iloc[i_tr], df.iloc[i_va]
# grupo intacto Y proporción de clases: StratifiedGroupKFold
```

MÉTRICAS ↑ mejor alto · ↓ mejor bajo

```
from sklearn.metrics import (roc_auc_score, average_precision_score,
                             f1_score, mean_squared_error, mean_absolute_error)

roc_auc_score(y, proba)          ↑ orden; 0.5 = azar
average_precision_score(y, proba) ↑ clase rara; base = % positivos
f1_score(y, pred, average="macro") ↑ cada clase pesa igual
np.sqrt(mean_squared_error(y, p)) ↓ RMSE, en unidades de y
mean_absolute_error(y, p)        ↓ menos sensible al outlier

# recall dentro de un presupuesto del 5 %
k = int(0.05 * len(proba))
top = np.argsort(-proba)[:k]
recall_k = y.iloc[top].sum() / y.sum()
```

Con 5 % de positivos, la accuracy es 95 % sin hacer nada.

MIRAR DENTRO DEL PIPELINE

```
model.named_steps["pre"]          # el ColumnTransformer
model[: -1].fit_transform(X).shape # cuántas columnas salen
model.named_steps["pre"].get_feature_names_out()[:10]

# validación cruzada, con el pipeline entero dentro
from sklearn.model_selection import cross_val_score
cross_val_score(model, X, y, cv=5, scoring="roc_auc").mean()
# scoring: "roc_auc" · "average_precision" · "f1_macro"
# "neg_root_mean_squared_error"
```

Las cinco reglas que salvan la nota

- El holdout es sagrado. Una predicción por fila, en el orden del archivo. Nunca drop_duplicates, dropna ni sort_values sobre él.
- fit sólo en el train. Si escribiste fit dos veces, está mal. Encoders, imputers, medianas y medias: se aprenden una vez.
- Lo del enunciado, fuera; lo aprendido, dentro. Convertir texto a número lo sabes antes de abrir el archivo → pandas. Qué categorías existen sólo lo sabes mirando → pipeline.
- handle_unknown="ignore" siempre. No cuesta nada y evita que un nivel nuevo en el holdout tumbte la tarea entera.
- Métrica sospechosamente buena = fuga. Busca la columna que se escribe después del desenlace. En una entrevista, ésa es la respuesta correcta.

CUANDO EL PIPELINE NO ARRANCA

MENSAJE	ARREGLO
could not convert string to float: 'X'	Esa columna va en CAT, dentro del OneHotEncoder
X has 22 features, but ... expecting 23 feature names should match those ... during fit	Codificaste las dos tablas por separado. Un solo encoder
Found unknown categories [...] during transform	handle_unknown="ignore"
Input X contains NaN.	HistGB, o SimpleImputer dentro del pipeline
Input y contains NaN	Filtra las filas sin target — sólo en el train

QUÉ COLUMNAS NO ENTRAN AL MODELO

El id	Identifica la fila. No generaliza a ninguna otra
Lo constante	nunique(dropna=False) <= 1: no aporta información
La copia	Una columna que es otra reescalada. Quédate con una
El futuro	Lo que se escribe después del desenlace (una oferta de retención, una fecha de alta). Existe en el histórico y estará vacío el día que el modelo corra

Ésa última es la pregunta de entrevista: «¿esta columna ya estaba escrita en el momento en que hay que predecir?». Si no, fuera.

OTROS MODELOS, EN UNA LÍNEA

```
from sklearn.linear_model import LogisticRegression, Ridge
from sklearn.ensemble import RandomForestClassifier

LogisticRegression(max_iter=1000) # rápido, interpretable,
RandomForestClassifier(n_estimators=300, random_state=0, # exige imputar y escalar
                      n_jobs=-1) # tampoco come NaN
Ridge(alpha=1.0) # regresión lineal regularizada
```

Con 90 minutos: HistGB primero. Los demás, sólo si el enunciado pide interpretabilidad o algo falla.

El reloj

0-8

Leer el enunciado **entero**. Cada defecto está escrito como una regla; subráyalos. Anota qué tarea depende de cuál.

8-20

Cargar, coercion tipos, contar lo que falta. Dejar las tareas fáciles ya cobradas.

20-45

Limpieza y features, una función por regla, probando cada una con cuatro filas.

45-75

El modelo. Un pipeline, hiperparámetros de fábrica. Validar. Sólo después, más features.

75-90

Releer el enunciado contra tus respuestas: tipos de retorno, número de filas, orden.

Se falla gastando cincuenta minutos en el EDA y ocho en el modelo. Califica pronto y a menudo.

CARGAR SIN ROMPER NADA

```
df = pd.read_csv(p, keep_default_na=False, na_values=[""])
# † si un valor legítimo se llama "None" o "NA" (un producto,
# una categoría), el default de pandas lo convierte en NaN

pd.to_numeric(s, errors="coerce")          # texto → número, resto NaN
pd.to_datetime(s, format="mixed", dayfirst=True)  # dos formatos
s.astype(str).str.replace(r"^[0-9.-]+", "", regex=True)
# † 'S/ 1,234.50' → '1234.50' (se lleva la coma de los miles)
```

Después de parsear fechas, mira siempre `min()` y `max()`. Sin `dayfirst`, 04/03/2026 se lee como 3 de abril: no hay error, hay miles de filas mal y un archivo que llega hasta diciembre cuando acababa en julio.

EDA EN SEIS LÍNEAS

```
df.shape; df.dtypes; df.head()
df.isna().sum()[lambda s: s > 0]          # sólo lo que falta
df.nunique(dropna=False)                  # constantes = 1, ids = n
df.describe(include="all").T              # min/máx: busca sentinelas
df["col"].value_counts(dropna=False)      # casing, niveles raros
df.duplicated(subset="id").sum()
```

Sentinelas: -1, 999999, un 0 imposible, 1900-01-01. Se ven en el `min/max`, nunca en el `isna`.

RECETAS DE LIMPIEZA

```
# sentinela → hueco de verdad
df["score"] = df["score"].replace(-1, np.nan)
df["ingreso"] = df["ingreso"].replace(999999, np.nan)

# mayúsculas y espacios: doblar ANTES de mapear
df["c"] = df["c"].astype(str).str.strip().str.lower().map(MAPA)

# una unidad distinta en un lote (céntimos por soles)
m = df["precio"] > 1000
df.loc[m, "precio"] = df.loc[m, "precio"] / 100

# fahrenheit sólo donde toca
f = df["unidad"] == "F"
df.loc[f, "temp"] = (df.loc[f, "temp"] - 32) * 5 / 9

# duplicados por id (no por fila entera: el 4º decimal cambia)
df = df.drop_duplicates(subset="id", keep="first")

# blanco que significa cero, no dato que falta
df["promo"] = pd.to_numeric(df["promo"], errors="coerce").fillna(0)

# columnas que no deben llegar al modelo
drop = ["id"] + [c for c in df if df[c].nunique(dropna=False) <= 1]
```

FEATURES QUE SUMAN

```
# ratio sin infinitos: el 0 se vuelve NaN antes de dividir
m = df["meses"]
df["gasto_mes"] = df["total"] / m.where(m > 0)

# fechas
d = pd.to_datetime(df["fecha"], format="mixed", dayfirst=True)
df["dow"] = d.dt.dayofweek
df["finde"] = (d.dt.dayofweek >= 5).astype(int)
df["dia"] = (d - d.min()).dt.days          # tendencia
df["mes_del_año"] = df["meses"] % 12      # renovación

# contra el grupo: aprendido en el TRAIN, aplicado a los dos
med = train.groupby("sku")["precio"].median()
for f in (train, holdout):
    f["precio_vs_sku"] = f["precio"] / f["sku"].map(med)

# orden explícito para una categórica ordinal
df["nivel"] = df["educacion"].map({"primaria":1, "secundaria":2,
    "tecnic":3, "universitaria":4, "postgrado":5})
```

Un árbol aproxima un cociente apilando cortes: dárselo hecho vale más que cien iteraciones de boosting.

AGRUPAR, UNIR, REFORMAR

```
# resumen por grupo
df.groupby("tienda")["units"].agg(["count", "sum", "mean"])
df.groupby(["tienda", "sku"], as_index=False)["units"].sum()

# transform: el valor del grupo pegado a CADA fila (no reduce)
df["media_t"] = df.groupby("tienda")["units"].transform("mean")
df["vs_media"] = df["units"] / df["media_t"]

# contar y ordenar categorías
df["c"].value_counts(dropna=False).head(10)
df["id"].duplicated(keep=False)          # marca TODAS las repetidas

# unir sin perder filas ni duplicarlas
out = df.merge(otro, on="id", how="left", validate="m:1")
print(len(out) == len(df))      # False = "otro" repite ids

# días ausentes = calendario menos lo que hay
cal = set(pd.date_range(d.min(), d.max(), freq="D"))
faltan = {g: len(cal - set(x)) for g, x in d.groupby(df["tienda"])} }
```

CUANDO LA CLASE ES RARA

```
y.value_counts(normalize=True)          # míralo ANTES de elegir métrica

# 1 · el modelo puede pesar más la clase rara
LogisticRegression(class_weight="balanced", max_iter=1000)
RandomForestClassifier(class_weight="balanced_subsample")

# 2 · o eliges el umbral tú, sobre la probabilidad
proba = model.predict_proba(X_va)[:, 1]
pred = (proba >= 0.30).astype(int)      # 0.5 no es sagrado

# 3 · si piden "los k que revisaremos", es un orden
k = int(0.05 * len(proba)); top = np.argsort(-proba)[:k]
```

SMOTE no está en scikit-learn y casi nunca hace falta. **Primero la métrica correcta, después el umbral.**

Antes de entregar

- `len(pred) == len(holdout)`, y en el orden del archivo
- `np.isfinite(pred).all()` — ni NaN ni inf
- Probabilidad → `predict_proba(X)[:, 1]`, no `predict`
- Regresión de cantidades → `np.clip(pred, 0, None)`
- Cada función devuelve **el tipo que pide el enunciado**: dict, list, DataFrame, tupla, array
- El holdout conserva todas sus filas y ninguna columna nueva sobra
- Releída la lista de defectos del enunciado, uno por uno

SI TE QUEDAN DIEZ MINUTOS

En este orden, y ni uno más:

- Que **todas** las funciones devuelvan algo válido, aunque sea mediocre: cada tarea se califica sola y media respuesta suma.
- Comprobar el `len` y el orden del holdout.
- Una feature de cociente o de calendario, que es donde está la mejora barata.
- No tocar hiperparámetros: de fábrica ya estás cerca del techo.