**Qué hace evalcards**

Genera un **reporte Markdown** con **métricas** y **gráficos** listos para:

* **Clasificación binaria** (con o sin probabilidades).
* **Clasificación multiclase (OvR)** con curvas **ROC/PR por clase** y **AUC macro OvR**.
* **Regresión**.
* **Forecasting** (series de tiempo) con **sMAPE (%)** y **MASE**.

Además:

* Guarda todo en ./evalcards\_reports (o lo que pases en --outdir).
* Se usa por **API Python** o **CLI**.

**Métricas y gráficos por tarea**

**Clasificación (binaria y multiclase OvR)**

* Métricas: accuracy, precision/recall/F1 (macro y weighted),  
  **binaria**: roc\_auc; **multiclase**: roc\_auc\_ovr\_macro.
* Gráficos: **matriz de confusión**, **ROC** y **PR**  
  (binaria: 1 par de curvas; multiclase: 1 par **por clase**).

**Regresión**

* Métricas: MAE, MSE, RMSE, R².
* Gráficos: **ajuste y vs ŷ** y **residuales**.

**Forecasting**

* Métricas: MAE, MSE, RMSE, **sMAPE (%)**, **MASE** (requiere season e idealmente insample).
* Gráficos: **ajuste y vs ŷ** y **residuales**.

**Ejemplos (API Python)**

**1) Clasificación binaria (scikit-learn)**

from sklearn.datasets import make\_classification

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

import numpy as np

from evalcards import make\_report

X, y = make\_classification(n\_samples=600, n\_features=10, random\_state=0)

X\_tr, X\_te, y\_tr, y\_te = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

clf = LogisticRegression(max\_iter=1000) # cualquier clasificador sirve

clf.fit(X\_tr, y\_tr)

y\_pred = clf.predict(X\_te)

# Probabilidades (para ROC/PR y AUC). Si usas SVM, activa probability=True.

proba = clf.predict\_proba(X\_te)[:, 1]

make\_report(y\_te, y\_pred, y\_proba=proba, path="rep\_bin.md", title="Clasificación binaria")

**2) Multiclase OvR (scikit-learn)**

from sklearn.datasets import load\_iris

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from evalcards import make\_report

X, y = load\_iris(return\_X\_y=True)

X\_tr, X\_te, y\_tr, y\_te = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

clf = RandomForestClassifier(random\_state=0)

clf.fit(X\_tr, y\_tr)

y\_pred = clf.predict(X\_te)

# Matriz de probabilidades (n\_samples, n\_classes)

proba = clf.predict\_proba(X\_te) # columnas en el orden clf.classes\_

make\_report(

y\_te, y\_pred, y\_proba=proba,

labels=[f"Clase\_{c}" for c in clf.classes\_],

path="rep\_multi.md", title="Multiclase OvR"

)

**3) Regresión (scikit-learn)**

from sklearn.datasets import make\_regression

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import train\_test\_split

from evalcards import make\_report

X, y = make\_regression(n\_samples=600, n\_features=8, noise=10, random\_state=0)

X\_tr, X\_te, y\_tr, y\_te = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

reg = RandomForestRegressor(random\_state=0).fit(X\_tr, y\_tr)

y\_pred = reg.predict(X\_te)

make\_report(y\_te, y\_pred, path="rep\_reg.md", title="Regresión")

**4) Forecasting (sMAPE/MASE)**

import numpy as np

from evalcards import make\_report

rng = np.random.default\_rng(0)

t = np.arange(360)

y = 10 + 0.05\*t + 5\*np.sin(2\*np.pi\*t/12) + rng.normal(0,1,360)

y\_train, y\_test = y[:300], y[300:]

# Supón que tu modelo produce y\_hat para el horizonte de test:

y\_hat = y\_test + rng.normal(0, 1.2, y\_test.size) # ejemplo

make\_report(

y\_test, y\_hat,

task="forecast", season=12, insample=y\_train,

path="rep\_forecast.md", title="Forecast"

)

**Ejemplos (CLI)**

**Binaria**

evalcards --y\_true y\_true.csv --y\_pred y\_pred.csv --proba y\_proba.csv --out rep\_bin.md

**Multiclase OvR** (un CSV con N columnas, una por clase)

evalcards --y\_true y\_true.csv --y\_pred y\_pred.csv --proba y\_proba\_multiclass.csv \

--class-names "A,B,C" --out rep\_multi.md

**Forecast**

evalcards --y\_true y\_true\_test.csv --y\_pred y\_pred\_test.csv \

--forecast --season 12 --insample y\_insample.csv \

--out rep\_forecast.md

**¿Para qué modelos de ML sirve?**

Evalcards **no depende del modelo**, sino de lo que **entregas**:

* **Clasificación**: cualquier modelo que produzca **etiquetas** (predict) y, si quieres curvas ROC/PR/AUC, **probabilidades** (predict\_proba) o scores calibrados.
  + Funciona con **scikit-learn** (LogisticRegression, SVC con probability=True, RandomForestClassifier, GradientBoostingClassifier), **XGBoost/LightGBM/CatBoost** (si usas predict\_proba), **Keras/PyTorch** (si generas tus probabilidades y etiquetas).
  + **Multiclase**: pasa y\_proba como **matriz (n\_samples, n\_classes)**. El orden de columnas debe corresponder al orden de tus clases (p.ej. clf.classes\_ en sklearn).
* **Regresión**: cualquier regressor (LinearRegression, Lasso/Ridge, RandomForestRegressor, GradientBoosting, XGBRegressor/LightGBM/CatBoostRegressor, redes neuronales…). Solo necesitas y\_true y y\_pred.
* **Forecasting**: cualquier enfoque que entregue un vector de **pronóstico** para el horizonte (ARIMA/SARIMA, Prophet, NeuralProphet, modelos de ML con lags, LSTM/Transformers, etc.). Para **MASE** idealmente provee insample (serie de entrenamiento) y **season** (p.ej. 12 para mensual con anualidad).

**Límites actuales (útiles para saber)**

* **No** soporta **multi-label** (varias etiquetas verdaderas por instancia).
* En **multiclase** calcula **AUC macro OvR** y curvas **por clase** (no micro/macro PR/ROC agregadas—se puede añadir más adelante).
* No calcula **métricas de ranking** (MAP/NDCG), ni métricas **calibración** (Brier) todavía.
* Forecasting: se centra en error de punto (sMAPE/MASE), **no** en métricas probabilísticas.