Manual COMPLETO – Equipo 2 (ML + Microservicio)

Qué es el módulo ML en esta solución y cómo encaja

El módulo ML prioriza candidatos a exoplanetas (1=candidato/confirmado, 0=falso positivo) a partir de catálogos Kepler/K2/TESS. Produce un modelo (Random Forest baseline) que se exporta a `ml/model.pkl` y se sirve vía un microservicio FastAPI. La App Django consume ese servicio por HTTP (`/predict` y `/explain`).

- Entrada mínima: `period_days`, `duration_hours`, `depth_ppm`, `teff_K`, `mag` (o payload NASA + `source`).
- Salida: `label` y `score` (probabilidad). Explicabilidad: `importancias` (top features).
- Diseñado para 24h: pipeline reproducible, artefactos simples, plan B si falla el servicio.

Herramientas y entorno (recomendado)

- Notebook: Google Colab o JupyterLab.
- Python 3.10+ con: `pandas`, `numpy`, `scikit-learn`, `matplotlib`, `joblib`.
- Validación visual rápida: **Orange** (GUI).
- Servicio: **FastAPI** + `uvicorn`. Publicación: **ngrok** (temporal), **Render/Cloud Run** (estable).
- Control de versiones: GitHub (opcional).

Datasets: columnas específicas y etiqueta (label)

- Kepler (KOI): `koi_period`, `koi_duration`, `koi_depth`, `koi_kepmag`, `koi_steff`, `koi_disposition`, `koi_pdisposition`.
- K2: `pl_orbper`, `st_teff`, `sy_gaiamag/sy_vmag/sy_kmag`, `disposition`. (Profundidad/duración suelen faltar).
- TESS: `pl_orbper`, `pl_trandep`, `st_teff`, `st_tmag`, `tfopwg_disp`.
- Unificación (min): `period_days`, `duration_hours`, `depth_ppm`, `teff_K`, `mag`, `label`.
- Etiquetas → `label`: Kepler(CANDIDATE/CONFIRMED=1; FALSE POSITIVE=0), K2(CONFIRMED/CANDIDATE=1; FALSE POSITIVE/REFUTED=0), TESS(PC/CP/KP/APC=1; FP/FA=0).

Limpieza de datos (paso a paso)

- Eliminar filas sin `period_days` o sin `label`.
- Convertir columnas a numéricas (`errors='coerce'`).
- Imputación por **mediana** en numéricas (o eliminar filas si la proporción de NaN es baja).
- Crear features derivadas: `log_period = log1p(period_days)`, `log_depth = log1p(depth_ppm)`, `z_teff` (si guardas media y std).
- Opcional (si hay tiempo): outliers vía percentiles (p1–p99) o winsorizar; correlación (`df.corr()`), eliminar colinealidad si afecta lineales.

```
# Esqueleto de limpieza
import numpy as np, pandas as pd
df = combined.dropna(subset=["period_days","label"]).copy()
df["period_days"] = pd.to_numeric(df["period_days"], errors="coerce")
df["duration_hours"] = pd.to_numeric(df["duration_hours"], errors="coerce")
df["depth_ppm"] = pd.to_numeric(df["depth_ppm"], errors="coerce")
df["teff_K"] = pd.to_numeric(df["teff_K"], errors="coerce")
df["mag"] = pd.to_numeric(df["mag"], errors="coerce")
```

```
df["log_period"] = np.log1p(df["period_days"])
df["log_depth"] = np.log1p(df["depth_ppm"].clip(lower=0))
teff_mean, teff_std = df["teff_K"].mean(), df["teff_K"].std(ddof=0)
df["z_teff"] = (df["teff_K"] - teff_mean) / teff_std
```

Partición, entrenamiento, validación y métricas

- Partición estratificada `train_test_split(test_size=0.2, stratify=label)`.
- Modelo baseline: **RandomForestClassifier** (n_estimators=400, min_samples_leaf=2).
- Métricas foco: **Recall** (para no perder candidatos) + **F1**. Reporta matriz de confusión y clasificación.
- Cross-val (si hay tiempo): StratifiedKFold(5) o `cross_val_score`.
- Clase desbalanceada: ajustar `class_weight='balanced'` si el recall es bajo (opcional).

```
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confus
ion_matrix, classification_report

num_cols = ["period_days","duration_hours","depth_ppm","teff_K","mag","log_period","log_dep
th","z_teff"]
X = df[num_cols].values
y = df["label"].astype(int).values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random
_state=42)

pipe = Pipeline([("imputer", SimpleImputer(strategy="median")),
                 ("clf", RandomForestClassifier(n_estimators=400, min_samples_leaf=2, rando
m_state=42))])
pipe.fit(X_train, y_train)
pred = pipe.predict(X_test); proba = pipe.predict_proba(X_test)[:,1]
print("ACC", round(accuracy_score(y_test,pred),3),
      "PREC", round(precision_score(y_test,pred),3),
      "REC", round(recall_score(y_test,pred),3),
      "F1", round(f1_score(y_test,pred),3))
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

Exportación del modelo e inferencia reproducible

- Exporta con `joblib.dump(pipe, 'ml/model.pkl')`.
- Guarda `columns.json` con el **orden exacto** de columnas usadas en entrenamiento.
- Crea un módulo de inferencia `ml/predict.py` con: `load_model()`, `predict_row(row)`, `top_features(row)`.

```
from joblib import dump
import json, os
dump(pipe, "ml/model.pkl")
json.dump({"columns": num_cols}, open("ml/columns.json","w"))

# ml/predict.py (esqueleto)
from joblib import load
import numpy as np, os, json
MODEL=None; COLS=None
```

```
def load_model():
    global MODEL
    if MODEL is None: MODEL = load(os.path.join(os.path.dirname(__file__),"model.pkl"))
    return MODEL
def predict_row(row:dict):
    m = load_model()
    p=float(row.get("period_days",0) or 0); d=float(row.get("duration_hours",0) or 0)
    depth=float(row.get("depth_ppm",0) or 0); teff=float(row.get("teff_K",0) or 0); mag=flo
at(row.get("mag",0) or 0)
    X = np.array([[p,d,depth,teff,mag,np.log1p(p),np.log1p(max(depth,0)),0.0]])
    prob = float(m.predict_proba(X)[0][1])
    return ("candidato" if prob>=0.5 else "no_candidato"), prob
```

Empaquetar como microservicio (FastAPI) y publicar

- Usa el ZIP **ml-microservicio-fastapi** (incluye `app.py`, `ml/predict.py`, `Dockerfile`, `Procfile`).
- Local: uvicorn; Colab: ngrok; Producción: Render/Cloud Run.
- Seguridad: `API_TOKEN` y header `Authorization: Bearer ...`.

```
# Colab + ngrok (temporal)
!pip -q install fastapi "uvicorn[standard]" joblib scikit-learn pydantic==2.8.2 pyngrok
from pyngrok import ngrok; public_url = ngrok.connect(8000, "http")
print(public_url); import nest_asyncio, uvicorn, os
os.environ["API_TOKEN"]="mys3cret"; nest_asyncio.apply()
uvicorn.run("app:app", host="0.0.0.0", port=8000)
```

Contrato de API (para la App)

- **POST /predict**: entrada **normalizada** o **NASA+source**; salida `{label, score}`.
- **POST /explain**: salida `{importancias:{...}}`.
- Cabecera: `Authorization: Bearer `.

```
# Normalizado
{"period_days":3.2,"duration_hours":2.5,"depth_ppm":500,"teff_K":5500,"mag":12.1}
# NASA + source
{"source":"Kepler","koi_period":3.2,"koi_duration":2.5,"koi_depth":500,"koi_steff":5500,"ko
i_kepmag":12.1}
```

MVP mínimo (ML) y plan de 24h

- EDA mínima: histogramas de `period_days` y `pl_trandep` (si disponible), scatter `period_days` vs `depth_ppm`.
- Baseline RF entrenado y exportado a `ml/model.pkl`.
- Servicio FastAPI corriendo y probado con 3 ejemplos.
- Documentar 2 casos: claro vs dudoso; matriz de confusión y métricas en una tabla simple.

Orange (opcional) – pasos exactos

- File → carga `demo_candidates_unified.csv`.
- Select Columns: Target=`label`; Features=numéricas; Meta=`source/id`.
- Impute (mediana) → Normalize → Test&Score; (5-fold) con RF/GB → Confusion Matrix/ROC.
- Captura pantallas para el pitch y compáralo con tus métricas en scikit-learn.

Checklist final (ML)

- Modelo exportado (`ml/model.pkl`) + `columns.json`.
- Microservicio responde `/health`, `/predict`, `/explain`. Token configurado.
- 3 requests de ejemplo guardados (Thunder/Postman).
- CSV demo y guía de payload para el Equipo 1 y Equipo 3.