**1) Problem framing**

* **Goal:** classify a Kepler Object of Interest (KOI) as **CANDIDATE (1)** or **CONFIRMED (0)** using the KOI “cumulative” table.
* **Labels (your counts):** CONFIRMED=2,746; CANDIDATE=1,979; FALSE POSITIVE=4,839 (dropped).
* **Inputs:** astrophysical + pipeline features (e.g., transit depth, duration, period; star Teff, radius, mass; SNR; CDPP; etc.).

**2) Data selection & target**

* Keep only rows with koi\_disposition ∈ {CONFIRMED, CANDIDATE}.
* Map target to integers: CONFIRMED → 0, CANDIDATE → 1.

**3) Preprocessing pipeline (exactly what the model sees)**

1. **Column pruning**
   * Drop identifiers/metadata that don’t carry physical signal (IDs, names, vetting dates).
   * Drop columns with **>40% missingness** (tunable) to avoid heavy imputation noise.
2. **Type split**
   * **Numeric features:** remain numeric.
   * **Categorical features:** low-cardinality categoricals are kept; very high-cardinality ones are dropped (likely IDs).
3. **Imputation & encoding**
   * **Numeric:** median imputation (robust to outliers) → optional standardization (trees don’t need scaling, but harmless).
   * **Categorical:** most-frequent imputation → **one-hot encoding** with handle\_unknown="ignore".
4. **Leakage guard**
   * All of the above is wrapped in a **scikit-learn Pipeline** and fit **inside CV folds** (and refit on train only for holdout), so statistics from the test fold never leak into training.

**4) Train/validation protocol**

* **Split:** 70/30 train/test (stratified).
* **Model selection:** **RepeatedStratifiedKFold 10×5** (50 folds) for robust estimates.
* **Score used to pick params:** **F1** (balances precision/recall).
* **Tuning grid (216 combos):**  
  n\_estimators ∈ {200,400,800}, max\_depth ∈ {None,10,20,30}, min\_samples\_split ∈ {2,5,10}, min\_samples\_leaf ∈ {1,2,4}, max\_features ∈ {sqrt,log2}.

**Your best parameters:**  
n\_estimators=400, max\_depth=30, max\_features='sqrt', min\_samples\_split=5, min\_samples\_leaf=1.

**Your CV results (means ± SD):**  
Accuracy **0.878 ± 0.014**, Precision **0.863 ± 0.022**, Recall **0.844 ± 0.025**, F1 **0.853 ± 0.017**.

**5) How Random Forest actually works (granular)**

A Random Forest (RF) is an ensemble of decision trees trained on perturbed views of the data:

1. **Bootstrap sampling**  
   Each tree is trained on a bootstrap sample (sample with replacement) of the training rows. This decorrelates trees.
2. **Feature sub-sampling at each split**  
   At every tree node, RF considers only a random subset of features of size max\_features.
   * With max\_features="sqrt" in classification, each split evaluates roughly √(p) features (where p is the number of input columns after preprocessing/one-hot).
   * This injects more randomness, reducing variance and improving generalization.
3. **Split criterion (Gini impurity)**  
   At a node with class proportions p0,p1p\_0, p\_1p0​,p1​, **Gini** is

G=1−∑kpk2=1−(p02+p12).G = 1 - \sum\_k p\_k^2 = 1 - (p\_0^2 + p\_1^2).G=1−k∑​pk2​=1−(p02​+p12​).

For each candidate feature/threshold, the algorithm computes the **impurity decrease**:

ΔG=G(parent)−(nLnG(L)+nRnG(R)),\Delta G = G(\text{parent}) - \left( \frac{n\_L}{n} G(L) + \frac{n\_R}{n} G(R) \right),ΔG=G(parent)−(nnL​​G(L)+nnR​​G(R)),

and chooses the split that maximizes ΔG\Delta GΔG, subject to constraints (min\_samples\_split, min\_samples\_leaf, max\_depth).

1. **Stopping rules**
   * max\_depth=30 caps the depth (your optimum).
   * min\_samples\_split=5 requires ≥5 samples to consider splitting.
   * min\_samples\_leaf=1 allows pure leaves when possible.
2. **Leaf outputs & probabilities**  
   A tree’s leaf stores the **empirical class probabilities** of training samples that reached it.  
   The forest **averages** those probabilities across trees:

P^(y=1∣x)=1T∑t=1TPt(y=1∣x),\hat{P}(y=1 \mid x) = \frac{1}{T}\sum\_{t=1}^T P\_t(y=1 \mid x),P^(y=1∣x)=T1​t=1∑T​Pt​(y=1∣x),

where TTT is the number of trees (n\_estimators=400).  
The **class prediction** is 1 if P^(y=1)≥τ\hat{P}(y=1) ≥ \tauP^(y=1)≥τ (default threshold τ=0.5\tau=0.5τ=0.5); otherwise 0.

1. **Why it works well here**
   * Strong, nonlinear interactions (e.g., between depth, duration, SNR, CDPP, and stellar parameters).
   * Robust across mixed scales and missingness (with imputation).
   * Less tuning-sensitive than boosted trees on this dataset size.

**6) Metrics you report (what they mean)**

* **Accuracy:** (TP+TN)/(TP+TN+FP+FN)(TP+TN)/(TP+TN+FP+FN)(TP+TN)/(TP+TN+FP+FN).
* **Precision:** TP/(TP+FP)TP/(TP+FP)TP/(TP+FP) — reliability of positive (candidate) predictions.
* **Recall (Sensitivity):** TP/(TP+FN)TP/(TP+FN)TP/(TP+FN) — how many true candidates you catch.
* **Specificity:** TN/(TN+FP)TN/(TN+FP)TN/(TN+FP) — how many non-candidates you correctly reject.
* **F1:** harmonic mean of precision and recall.

Because false positives/negatives have different scientific costs, you may prefer tuning the **decision threshold** τ\tauτ to trade precision vs. recall (see §8).

**7) Why those hyperparameters matter**

* **n\_estimators (400):** more trees → lower variance → smoother probability estimates (diminishing returns after a few hundred).
* **max\_depth (30):** allows complex boundaries while limiting overfitting.
* **max\_features='sqrt':** stronger tree decorrelation → better ensemble generalization.
* **min\_samples\_split/leaf (5/1):** controls node purity and overfitting; your best result allows pure leaves but requires at least 5 samples to split.

**8) Threshold selection (operating point)**

Your Streamlit app defaults to τ=0.5\tau=0.5τ=0.5. You can set τ\tauτ to match your scientific goal:

* **Maximize F1:** search τ∈[0,1]\tau\in[0,1]τ∈[0,1] on CV predictions to maximize F1.
* **High precision regime:** raise τ\tauτ to reduce false positives.
* **High recall regime:** lower τ\tauτ to catch more candidates (accept more false positives).  
  Optional: **calibration** (Platt scaling or isotonic) can make P^(y=1)\hat{P}(y=1)P^(y=1) better reflect empirical frequencies if you need calibrated probabilities.

**9) Interpretability**

* **Gini importance (mean decrease in impurity):** gives a quick ranking of influential features but can be biased toward high-cardinality one-hot features.
* **Permutation importance:** measure drop in performance when shuffling a feature—more reliable across encodings.
* **SHAP values:** local, direction-aware explanations (which features pushed a specific KOI toward candidate vs. confirmed).

Typical high-signal features in exoplanet vetting include **transit depth**, **duration**, **period**, **SNR**, **impact parameter**, **radius ratio (Rp/R★)**, and **stellar properties** (e.g., Teff, radius, mass). Your importances/SHAP will make this concrete for your specific fit.

**10) Robustness & pitfalls**

* **Data leakage:** avoided via Pipeline + fit inside CV folds.
* **Class drift:** KOI curation is stable historically, but if definitions or upstream vetting change, retrain.
* **One-hot explosion:** many categoricals can inflate dimensionality; you already filtered high-cardinality IDs.
* **Imbalanced costs:** the dataset’s 0/1 counts are not extreme, but **scientific cost** asymmetry might still justify threshold or class-weight adjustments.

**11) Deployment path (your app)**

* **Input handling:** the app builds a row with your schema’s column order; missing fields are left None and imputed by the saved pipeline.
* **Prediction:** predict\_proba → probability of **candidate**; compare to user-chosen τ\tauτ.
* **Batch mode:** CSV upload, missing columns are added as None, outputs include probability + label.

**12) Reproducibility checklist**

* Fix random seeds (you did: 42).
* Pin library versions (you did in requirements.txt).
* Save the **entire pipeline** (preprocessor + RF), not just the classifier.
* Record the **best params** and CV protocol (10×5 RS k-fold with F1 selection).
* Keep the **feature schema** JSON alongside the model.

**TL;DR (how a single prediction is produced)**

1. Your raw inputs are aligned to the schema (missing allowed).
2. The pipeline imputes numerics (median), imputes categoricals (mode), one-hot encodes categoricals, and passes the resulting vector to RF.
3. 400 trees, each trained on bootstrapped data and random feature subsets, vote a probability for “candidate.”
4. The forest averages those probabilities; if ≥ threshold (default 0.5), the KOI is labeled **CANDIDATE (1)**, else **CONFIRMED (0)**.