

FRAMEWORK

1. High-level Instruction (Motivation → Process → Conclusion)

1.1 Motivation: Why Choose the KOI Dataset and `koi_score`

Task Background

The Kepler mission produced a large number of KOIs (Kepler Objects of Interest), where each KOI represents a signal that "looks like a planetary transit."

These signals include:

- Actual planet candidates
- Various false positives (binary stars, background stars, noise, etc.)

The Role of `koi_score`

The Kepler team uses an automated vetting system (**Robovetter**) to classify each KOI as:

- Candidate / False Positive
- And assigns a `koi_score` between 0–1, indicating how "reliable" the classification is



What is `koi_score` ?

A **quality/confidence metric** that integrates numerous tests and Monte Carlo experiments. In subsequent planet statistics and occurrence rate studies, it is often used with a threshold (e.g., ≥ 0.9) to select "high-quality samples."

Why Choose This Dataset and Target

Dataset Advantages:

- The KOI cumulative table is a **public, real-world, moderately-sized, multi-variable engineering dataset**
- Ideally suited for the course requirements of PCA + ML

Target Significance:

- `koi_score` is not a simple physical quantity, but rather a "quality score output from a complex pipeline," which closely aligns with the *Advanced Statistical Approaches to Quality* theme
- Computing it requires the full Robovetter suite plus extensive injection tests, making it non-reproducible and costly for ordinary users



Core Question of This Project

Can we use only the physical and observational features available in the KOI table, and apply statistical & machine learning methods to **classify KOIs into high-quality vs low-quality classes** based on a binary label derived from `koi_score`, creating a lightweight "quality assessment classifier"?

1.2 Process: EDA + PCA → Binary Classification with PyCaret

Target Problem (High Level)

Given the multi-dimensional features of KOIs (orbital period, transit depth, duration, SNR, stellar parameters, etc.), first perform standardization and exploratory data analysis (EDA), then use PCA (via the `pca` Python library) to extract key principal components. Use the continuous `koi_score` to construct a **binary label** by applying double thresholds (0.9 and 0.1), excluding middle-range samples. Train and compare multiple classification models using **PyCaret's classification module** on both the original feature space and the PCA feature space, focusing on **Logistic Regression, Random Forest, and MLP**, to determine which combination is best suited as a "lightweight quality classifier."

The overall workflow draws from the sample project: first perform **data cleaning + standardization + binary label construction + PCA visualization**, then conduct **systematic model comparison using PyCaret**, and finally interpret and discuss the results.

Step 1 – Data Acquisition, Cleaning, and Exploratory Data Analysis (EDA)

Step 1.1 Data Acquisition and Field Selection (KOI Columns to Retain)

Download the KOI table from NASA Exoplanet Archive (with complete header). After reading it in, **don't throw everything directly into the model**, but rather categorize fields into 4 functional groups:

(a) Identifier Fields (For Indexing Only, Not Features)

These columns are only used to label each KOI record and **do not enter the classification model**:

- `kepid` – Kepler Catalog ID
- `kepoi_name` – KOI Name
- `kepler_name` – Official Kepler Planet Name (if any)

These fields can be used to trace back to specific KOI individuals in the original table during subsequent analysis/plotting.

(b) Source for Binary Label Construction

`koi_score` – Disposition Score

- This is a continuous score in [0,1] representing vetting classification confidence
- **Role in this project:** Used to **derive the binary label** via thresholding, but **NOT used as a feature** in any model
- **Label construction:**
 - **High-quality KOIs (label = 1):** `koi_score ≥ 0.9`
 - **Low-quality KOIs (label = 0):** `koi_score ≤ 0.1`

- All KOIs with `0.1 < koi_score < 0.9` are excluded from the dataset (neither in training nor testing)



Important: `koi_score` itself is **not included as a predictor** in X. It is only used to create the binary target variable y. This avoids label leakage and ensures the model learns from physical/observational features alone.

(c) Labels and False Positive Flags for Analysis/Grouping Only (Not in Feature Matrix X)

These fields are used for "slicing/comparing" in EDA and results discussion, but **not as model predictors** to avoid label leakage:

Classification Result Types:

- `koi_disposition` – Exoplanet Archive Disposition
- `koi_pdisposition` – Disposition Using Kepler Data

False Positive Flags:

- `koi_fpflag_nt` – Not Transit-Like False Positive Flag
- `koi_fpflag_ss` – Stellar Eclipse False Positive Flag
- `koi_fpflag_co` – Centroid Offset False Positive Flag
- `koi_fpflag_ec` – Ephemeris Match Indicates Contamination False Positive Flag



These fields reflect the Robovetter/pipeline's own judgment results or intermediate states. If used to predict the quality label, they would highly overlap with the target. Therefore, they are only used in **description and visualization**, not included in the PCA or classification models' feature vectors.

(d) Physical and Observational Features Entering PCA and Classification (12-D Feature Vector X)

After cleaning the KOI table, we focus on a subset of continuous attributes that describe the transit signal, the planet-star system, and the overall observation quality.

These attributes form the feature vector \mathbf{X} for PCA and for all classification models in PyCaret.

For clarity we group them into four categories and give a short, non-technical explanation of each variable.

(1) Transit Geometry & Signal Quality

These quantities describe **how the light curve looks during a transit** and how strong the signal is in the data.

- **koi_period** – Orbital period [days]

Time between two successive transits of the same object. Shorter periods mean more frequent transits in the Kepler time series, while very long periods produce only a few events. The period affects how confidently the pipeline can detect and confirm a repeating signal.

- **koi_impact** – Impact parameter

A dimensionless number (roughly between 0 and 1) that tells how centrally the planet crosses the stellar disk.

Values near 0 correspond to a central transit; values near 1 correspond to a grazing transit that only clips the edge of the star. Unusual or extreme impact parameters can be a sign of an eclipsing binary or a poorly fitted signal.

- **koi_duration** – Transit duration [hours]

Length of a single transit event from ingress to egress. It is determined by the orbital period, the size of the star and planet, and the geometry of the orbit. Inconsistent or extreme durations relative to the period can indicate that the signal is not a clean planetary transit.

- **koi_depth** – Transit depth [ppm]

Relative drop in the star's brightness during transit, measured in parts per million. Deeper transits usually correspond to larger companions or stellar eclipses, while very shallow transits can be close to the noise level. The depth strongly influences how "planet-like" a candidate appears.

- **koi_model_snr** – Transit model signal-to-noise ratio

Ratio between the fitted transit signal and the noise level in the light curve. A higher SNR means a cleaner, more convincing detection, whereas low-

SNR events are more ambiguous and more likely to be classified as false alarms.

(2) Planet Properties & Irradiation

These variables describe **the size of the candidate planet and how much radiation it receives from the star**.

- `koi_prad` – Planetary radius [Earth radii]

Estimated radius of the candidate planet in units of Earth radii, inferred from the transit depth and the size of the host star. Extremely large radii are often associated with stellar companions rather than planets.

- `koi_teq` – Equilibrium temperature [K]

Approximate temperature the planet would have if it were a dark body in balance with the stellar radiation it receives (assuming a simple energy balance model). It combines information about the star's luminosity and the orbital distance, and helps distinguish very hot, close-in candidates from cooler, wider-orbit systems.

- `koi_insol` – Insolation flux [Earth flux]

Incident stellar radiation at the planet's orbit, expressed as a multiple of the solar flux received by Earth. Very high or very low insolation values reflect extreme orbital environments and can be correlated with how reliably the transit is detected.

(3) Stellar Properties

These quantities describe **the host star itself**, which strongly influences the appearance of the transit.

- `koi_steff` – Stellar effective temperature [K]

A measure of the star's surface temperature, related to its spectral type (e.g., cooler K/M stars vs. hotter F/G stars). It affects the star's luminosity and thus the expected transit depth for a given planet size.

- `koi_slogg` – Stellar surface gravity [$\log_{10}(\text{cm/s}^2)$]

Logarithm of the acceleration due to gravity at the star's surface. Dwarf stars and giant stars have very different surface gravities; giants tend to produce wider and deeper eclipses. This parameter is useful for distinguishing main-sequence planet hosts from evolved stars that often produce astrophysical false positives.

- `koi_srad` – Stellar radius [Solar radii]

Radius of the star in units of the Sun's radius. Together with transit depth it determines the inferred planet size. Very large stellar radii often imply that a deep transit is more likely due to another star than to a planet.

(4) Brightness / Observation Conditions

- `koi_kepmag` – Kepler-band magnitude

Apparent brightness of the host star in the Kepler photometric band, on an astronomical magnitude scale (smaller numbers mean brighter stars).

Brighter stars allow higher-precision photometry and typically yield higher signal-to-noise transits, which can affect the quality classification.

The above 12 attributes are treated as the **final feature set X**. They capture the most relevant physical and observational information for classifying KOI quality and are used in all PCA and classification experiments.

Excluded Columns: `ra`, `dec`, and `koi_time0bk`

The original KOI table also includes:

- `ra` – Right ascension (sky coordinate, in degrees)
- `dec` – Declination (sky coordinate, in degrees)
- `koi_time0bk` – Transit epoch [BKJD], i.e., the reference time of a particular transit

However, these three columns are **not** included in the feature vector X:

- `ra` and `dec` (**sky position**)

These angles only specify where the target lies on the celestial sphere within the Kepler field. They do not change the physical properties of the star–planet system or the shape of the transit. The Kepler pipeline processes the entire field in a uniform way, so the celestial coordinates are not expected to carry meaningful information about how "planet-like" a signal looks. To avoid adding extra dimensions that mainly describe pointing and sky location, we exclude `ra` and `dec` from PCA and all classification models.

- `koi_time0bk` (**transit epoch**)

This value sets the zero-point in time for the transit (when a particular transit occurs in the light-curve timeline). Shifting the epoch changes only *when* the transit happens, not *what* it looks like. It is therefore not physically

linked to the reliability of the signal. For model simplicity and interpretability, `koi_time0bk` is also dropped from the feature set.

In summary, all subsequent data analysis, PCA, and classification in this project are performed on a fixed **12-dimensional feature vector X** composed of the attributes listed in Sections (1)–(4) above. The columns `ra`, `dec`, and `koi_time0bk` are treated purely as meta-data and are not used as predictors.

(e) Strategy for Uncertainty Columns (`_err1`, `_err2`) in This Project

For upper/lower bound uncertainty columns appearing in the header, such as:

```
koi_period_err1, koi_period_err2, koi_duration_err1, koi_duration_err2, koi_depth_err1, koi_depth_err2,  
koi_prad_err1, koi_prad_err2, koi_teq_err1, koi_teq_err2, koi_insol_err1, koi_insol_err2, koi_steff_err1,  
koi_steff_err2, koi_slogg_err1, koi_slogg_err2, koi_srad_err1, koi_srad_err2
```

Processing Strategy:

- **Not retained**
 - In the main experiment ("PCA + PyCaret classification"), to avoid dimension explosion and further aggravating collinearity, the **baseline only uses central value columns as features**
-

Step 1.2 Basic Data Inspection and EDA

Basic Data Inspection

Use `df.info()`, `df.describe()`, `df.duplicated().sum()` to check:

- Row count, column count, data types
- Whether duplicate rows exist
- Whether any variables have severe missingness

Visualization:

- Histogram / KDE of `koi_score` to observe its distribution (expected to show two peaks near 0 and 1)
- This bimodal distribution justifies the threshold-based binary labeling approach
- Boxplots / violin plots of numerical features to check for outliers
- Correlation matrix heatmap to examine multi-collinearity among features

Binary Label Construction:

- Apply double thresholds to `koi_score`:
 - Label = 1 (high-quality): `koi_score ≥ 0.9`
 - Label = 0 (low-quality): `koi_score ≤ 0.1`
 - Exclude all samples with `0.1 < koi_score < 0.9`
- This creates a clean separation between high-quality and low-quality KOIs

Feature Selection and Label Leakage Control:

- Independent variables X: Only use observational/physical features, not `koi_score` or classification results
- Dependent variable y: Binary label derived from `koi_score`
- "Classification label" fields like `koi_disposition` / `koi_pdisposition` and false positive flags: **Not used as model input features** to avoid label leakage; only used as auxiliary variables in EDA and results discussion

Missing Values and Outlier Handling:

- Remove samples with missing `koi_score` (cannot construct label).
- For the selected numerical astrophysical features (the 12-dimensional feature vector X), do **not** apply global mean/median imputation. Instead, **remove any samples that have missing values in these features**.
- This strategy is feasible because, even after applying the double-threshold on `koi_score`, the dataset still contains on the order of ~3,600 low-quality (label 0) and ~2,900 high-quality (label 1) KOIs, so dropping a small fraction of rows with missing values does not threaten sample size or class balance.
- This also avoids fabricating astrophysical quantities (e.g., radii, insolation, periods) using global statistics, which could be physically misleading.
- Reasonably handle extreme outliers (delete or winsorize as appropriate, but avoid over-cleaning)

Data Splitting and Standardization:

Divide labeled data into train / test sets:

- Training set 70% (for model fitting via PyCaret)
- Test set 30% (for final evaluation)

Use `StandardScaler` for all numerical features:

- `fit` on the training set, then `transform` train/test with the same scaler
 - Ensure PCA and subsequent models all work in the standardized space
 - This produces `X_std` (standardized original features)
-

Step 2 – PCA (Principal Component Analysis) and Principal Component Visualization

Perform PCA Using the `pca` Python Library

We use the **Python `pca` library** (the dedicated PCA package) to compute principal components and generate visualizations:

- First fit all principal components without limiting the number
- Calculate the **explained variance ratio** for each principal component
- The `pca` library is used to:
 - Compute principal components
 - Generate scree plots, biplots, PC1–PC2 scatter plots, and loading plots

All numerical PCA results and visualizations in this project are obtained from this library implementation.

Determine Number of Principal Components to Retain

Plot explained variance and cumulative explained variance curves:

- Look for an "elbow" around 80%–90% cumulative variance to determine appropriate `k`
- Select the first `k` principal components as the PCA feature space, denoted as `Z_pca` (or `PC1, PC2, ..., PCk`)

Principal Component Space Visualization

Scatter Plot Visualization:

- Plot all KOIs on the (PC1, PC2) plane, using the binary label or `koi_disposition` as color labels
- Intuitively observe: whether high-quality and low-quality KOIs show structural differences in the principal component space

Feature Projection Visualization (Loadings/Biplot):

- Show which original features have large weights on PC1, PC2
- Interpret from a physical perspective what combination each principal component roughly represents (e.g., "geometry/SNR component," "stellar scale/temperature component," etc.)

Generate Two Feature Spaces

Obtain two versions of feature matrices for classification:

- **Standardized original features:** `X_std` (12 features)
- **PCA principal component features:** `z_pca` (retaining only the first `k` principal components, typically `k` chosen to explain ~80–90% variance)

PyCaret classification will be applied to both feature spaces.

Step 3 – Binary Classification Using PyCaret

PyCaret Classification Setup

All model training and comparison are now done with **PyCaret's classification module**.

There is **no separate, hand-coded set of scikit-learn models**; instead, PyCaret systematically compares multiple classifiers.

PyCaret is run **twice**, on the two feature spaces:

3.1 Classification on Original Standardized Features (`X_std`)

- Use `pycaret.classification` with `X_std` + binary label
- Call `setup()` to initialize the classification environment
- Call `compare_models()` to evaluate multiple classifiers on this feature set
- **Primary optimization metric: F1-score**
 - PyCaret's model comparison should be sorted/optimized by F1
 - F1 balances precision and recall for the positive class (high-quality KOs)
- **Secondary metrics:** Accuracy and ROC-AUC

3.2 Classification on PCA Features (`Z_pca`)

- Use the same binary label with the PCA scores as inputs
- Run a separate `setup()` + `compare_models()` to evaluate classifiers on the PCA-transformed features
- Use the same optimization metric (F1-score) for consistency
- Record Accuracy and ROC-AUC as well



Important: PyCaret's internal `pca` option should **not** be used. We perform PCA ourselves with the `pca` package to control and interpret the transformation explicitly.

3.3 Focus Algorithms for Detailed Analysis

Even though PyCaret trains many models, the **report's ML theory section** will focus on **three algorithms**:

1. **Logistic Regression (LR)** – Linear baseline, interpretable, suitable for understanding feature contributions
2. **Random Forest (RF)** – Tree-based ensemble, non-linear, provides feature importance
3. **Multi-Layer Perceptron (MLP)** – Neural network for non-linear decision boundaries, can capture complex patterns

These three algorithms should:

- Be introduced conceptually (what they do, key ideas, strengths/weaknesses)
- Be explicitly connected to this project's task: classifying **high vs low quality KOs**
- In the results section, be identified as either top-performing models from PyCaret or representative models chosen for deeper analysis

Evaluation Metrics for Selected Models

For the final selected models (LR, RF, MLP in each feature space), also present:

- **Confusion matrix**
- **ROC curves**

- Optionally, **Precision** and **Recall** (especially for the positive/high-quality class)
-

Step 4 – Comprehensive Comparison and Interpretation

Results Summary and Visualization

Organize all experimental results into tables and figures:

- **Rows:** Multiple models from PyCaret (focusing on LR, RF, MLP)
- **Columns:** Original features vs PCA features, F1-score, Accuracy, ROC-AUC, etc.
- Draw comparison plots (e.g., bar charts / tables) to help intuitively compare the performance of different models and feature spaces

Model Comparison

Compare model performance:

- **Across models:** Which algorithm (LR, RF, or MLP) performs best?
- **Across feature spaces:** Original (`x_std`) vs PCA (`z_pca`)
- Based on **F1-score as primary metric**, with Accuracy and ROC-AUC as secondary

Discussion of Best Model and PCA Value

Answer the following questions:

- Which model + feature space combination performs best overall in classifying KOI quality?
- Is PCA's impact on overall performance and model stability positive or negative?
- How does PCA affect LR, RF, and MLP differently?
- Considering actual deployment costs (training time, model complexity), which combination is most suitable?

Interpretation of Principal Components and Original Features

Utilize:

- Random Forest feature importance (from `X_std` experiments)
- PCA loadings (from `pca` library output)
- Logistic Regression coefficients (if applicable)
- If necessary, use SHAP / permutation importance for interpretability analysis of the best model

Discuss:

- Which original features or principal components are most discriminative for quality classification
- Their rough astrophysical meanings (e.g., "high SNR + reasonable geometry" corresponds to high-quality KOIs, etc.)

Return to Task Motivation

Summarize the project's conclusions:

- Using EDA + PCA + PyCaret classification, how well can we separate high-quality from low-quality KOIs?
 - Can these models serve as a "lightweight, interpretable" auxiliary tool for pre-screening KOIs?
 - What limitations exist and future improvement directions (more features, time series, ensemble methods, etc.)
-

1.3 Conclusion: Report and Deliverables

Final Deliverables Include:

1. Jupyter Notebook (on GitHub)

- Complete code from data loading, preprocessing, binary label construction, PCA (using `pca` library), PyCaret classification to results visualization
- Clear markdown explanations for easy reproduction by the instructor

2. 7–10 Page Two-Column PDF Report

- Use IEEE two-column format or similar
- Report cover page includes:
 - Your name

- Student ID
- GitHub repository link

3. Complete Project Narrative in Report

- Why choose KOI dataset and binary quality classification (Motivation)
- How to design binary label construction + PCA + PyCaret classification pipeline (Process)
- Model performance comparison and significance for "quality assessment" (Conclusion)
- Focus on F1-score, confusion matrices, and ROC curves in results section