

# Phase 1: Data Loading and Summary

The **first step** in the exoplanet habitability classification pipeline is handled by the `ExoplanetDataLoader` class, located in `src/data/load_data.py`. This class is responsible for:

## ✓ Responsibilities:

- **Validating the file path** and ensuring a `.csv` file is provided.
- **Loading the dataset** from a CSV file while optionally skipping commented lines (common in astronomical datasets).
- **Summarizing the dataset**, including:
  - Data types and non-null counts
  - Missing value counts per column
  - Basic statistical summaries (mean, std, min, max, etc.)

Class: `ExoplanetDataLoader`

## Methods:

- `load()`: Reads the CSV and returns a `pandas.DataFrame`. Automatically skips commented metadata lines if `skip_comments=True`.
- `summarize()`: Prints:
  - Data types and non-null entries
  - Missing value counts per column
  - Descriptive statistics (via `.describe()`)
- `get_feature_target_split(target_column: str)`: Returns a tuple (X, y) of features and the target column.

## Phase 2: Statistical Exploration of Exoplanet Features

The second stage of the pipeline is a scientific exploration of the raw feature space via the `ExoplanetStatisticalExplorer` class. This module performs elite-level statistical diagnostics, data distribution inference, and correlation structure mapping, all grounded in robust numerical science. The goal is to drive **preprocessing recommendations**, dimensionality reduction, and feature interpretability from an **astrophysical** and **statistical** lens.

### Class Overview: `ExoplanetStatisticalExplorer`

This class orchestrates a detailed **Exploratory Data Analysis (EDA)** pipeline that includes:

1. **Distributional Diagnostics**
2. **Multi-modal Correlation Analysis**
3. **Preprocessing Recommendations**

It is fully compatible with exoplanetary datasets from the NASA Exoplanet Archive and can be extended to similar astrophysical feature sets.

#### 1. Distributional Diagnostics

The `_distribution_analysis()` method examines the statistical shape (skewness and kurtosis) of critical physical features such as:

- Planet Radius (`pl_rade`, `pl_radj`)
- Planet Mass (`pl_bmasse`, `pl_bmassj`)
- Orbital Parameters (`pl_orbper`, `pl_orbsmax`)

- Stellar Properties (st\_teff, st\_mass, st\_rad, st\_met)
- Insolation and Equilibrium Temperature (pl\_insol, pl\_eqt)

### Transformation Logic

Using empirical thresholds from statistical theory:

- **Log Transform** is suggested for highly positively skewed, strictly positive distributions.
- **Yeo-Johnson Power Transform** is applied for moderate skewness or mixed-sign variables.
- **Winsorization** is recommended for extreme outliers (high kurtosis).
- **Standard Scaling** is always recommended for numerical comparability.

Each recommendation is stored in a structured dict:

### Visualization

Histograms and density plots for all target features are automatically generated and saved, enabling visual inspection of the impact of preprocessing.

## 2. Multi-Level Correlation Structure

The `_correlation_structure()` method calculates **three distinct correlation matrices** to capture both linear and non-linear relationships:

Correlation Type	Method Used	Use Case
------------------	-------------	----------

Pearson	<code>df.corr('pearson')</code>	Linear dependencies
Spearman	<code>df.corr('spearman')</code>	Monotonic relationships
Distance	<code>dcor.distance_correlation(x, y)</code>	Arbitrary nonlinear interactions

### Example Features Analyzed

- Stellar Temperature vs. Planet Insolation
- Planet Equilibrium Temperature vs. Orbital Period
- Metallicity vs. Planet Mass

## 3. Preprocessing Recommendation Report

After analysis, the `explore()` method prints an elite-level, human-readable summary for preprocessing strategy:

This phase creates a scientifically justified preprocessing map for each feature to guide normalization, transformation, and scaling for downstream pipelines.

### Output Summary

After execution, this phase yields:

- Dictionary of transformation suggestions
- Three correlation matrices (pearson, spearman, distance)
- Distribution plots and correlation heatmap

## Phase 3 ExoplanetFeatureEngineer

The `ExoplanetFeatureEngineer` class is the intellectual engine of this habitability classification pipeline. It is designed with astrophysical rigor, scientific explainability, and production-grade modularity. It performs feature synthesis from raw astrophysical measurements to create high-signal features tailored for ML models trained to assess exoplanet habitability.

### Scientific Design Philosophy

This component embodies the following elite principles:

- **Astrophysics-driven ML:** Features are derived from first principles and leading literature in astrobiology, stellar physics, planetary science, and habitability theory (e.g., Kopparapu 2013, Ida & Lin 2004, Barnes 2017).
  - **Explainability-by-construction:** Many features are not only biologically plausible but interpretable by domain experts — crucial for trust and transparency.
  - **Pipeline architecture:** Feature transformations are modularized using `scikit-learn`'s `Pipeline` API to enable integration, reproducibility, and deployment readiness.
  - **Robust validation:** Every transformation assumes nothing — input DataFrames are explicitly validated for required columns and scientific sanity (e.g., radius cannot be zero, stellar mass must be positive).
- 

### Feature Engineering Modules (Internals)

#### 1. `_validate_inputs(df)`

A precondition enforcement step ensuring that the DataFrame includes all scientifically required fields, such as:

- **Planetary characteristics:** `pl_rade`, `pl_bmasse`, `pl_eqt`, `pl_insol`, `pl_orbper`
- **Stellar environment:** `st_teff`, `st_met`, `st_mass`

- **System identifiers:** `hostname`, `pl_name`

Sanity checks go beyond presence — full-null columns or physically implausible values are flagged with actionable errors.

## 2. `_core_habitability_features(df)`

Implements canonical astrobiological thresholds:

- **Terrestrial classification** (`is_terrestrial`): Planets  $\leq 1.6$  Earth radii, referencing the radius valley.
- **In habitable zone** (`in_hz`): Receives stellar flux in the conservative range [0.36, 1.1] Earth flux units.
- **Temperature habitability** (`temp_habitable`): Equilibrium temperature between 273K and 373K.

Each indicator is a binary flag, serving as direct interpretable inputs and as targets for downstream classification calibration.

## 3. `_stellar_context_features(df)`

Encodes the astrophysical context around the host star:

- **Spectral type** (`spectral_type`): Derived from effective temperature using classification schemes from Pecaut & Mamajek.
- **Main sequence lifetime** (`ms_lifetime_gyr`): Estimated via a power law ( $\sim M^{-2.5}$ ) to assess evolutionary viability of the host.
- **Metallicity class** (`metallicity_class`): Binned into zones impacting planetary formation probability.
- **Core accretion probability** (`core_accretion_prob`): A sigmoid approximation from [Ida & Lin 2004] encoding the likelihood of terrestrial planet formation.

These features embed prior knowledge into the dataset, helping models focus on biologically and physically plausible systems.

#### 4. `_advanced_astrophysical_features(df)`

Derives high-level proxies for long-term habitability:

- **UV flux** (`uv_flux`): Estimated as a fourth-power function of stellar temperature scaled by flux, following Stefan–Boltzmann relations.
- **Tidal locking probability** (`tidal_lock_prob`): A coarse inverse-square proxy tied to orbital period and stellar mass.
- **Jeans parameter** (`jeans_parameter`): Represents likelihood of atmospheric escape, using gravitational binding vs. thermal energy balance.

These require unit conversions, clamping, and careful handling of edge cases. Every computation is fault-tolerant and fallbacks to statistical imputation where necessary.

#### 5. `_xai_optimized_features(df)`

Crafted for model transparency and human interpretability:

- **Earth Similarity Index (ESI)** (`esi`): A normalized composite distance from Earth based on radius and flux.
- **System complexity** (`system_complexity`): Logarithmic count of planets in the same stellar system, hinting at dynamical stability.

These features were selected for their semantic clarity in human-in-the-loop environments and XAI pipelines.

### Pipeline Integration — `_create_sk_pipeline()`

The entire feature engineering logic is encapsulated as a `scikit-learn` pipeline:

#### `_classify_habitability(df)`

Though not exposed in the pipeline by default, this internal method enables rule-based habitability tagging for rapid prototyping and data augmentation. It combines derived features to assign binary `habitable` labels based on astrophysical reasoning.

## References

- Kopparapu et al. (2013): Habitable Zones
- Fulton et al. (2017): Radius Valley
- Ida & Lin (2004): Core Accretion Theory
- Barnes (2017): Tidal Locking Dynamics
- Schulze-Makuch et al. (2011): Earth Similarity Index
- Rugheimer et al. (2015): UV Habitability Zones

Gough (1981): Stellar Lifetimes

Pecaut & Mamajek (2013): Spectral Classification

## Phase 2: Feature Engineering – **ExoplanetDataPreprocessor**

This stage is responsible for ingesting and preparing the NASA Exoplanet Archive dataset, an astrophysically rich dataset that captures characteristics of exoplanets and their host stars. The goal is to structure and verify the input data before deeper statistical modeling begins.

The **ExoplanetDataPreprocessor** class is the heart of the data preparation pipeline. This elite preprocessing engine is engineered for high-stakes astrostatistical classification. Below is a comprehensive walkthrough that explains the logic and scientific foundations behind this processor as one would deliver to chief scientists at MIT, Apple, or XAI.

### Class Overview

This class encapsulates:

- Domain-informed missing value imputation
- Robust transformations (log, Yeo-Johnson)
- Outlier resistance via Winsorization
- Feature-wise pipelines using **ColumnTransformer**
- Scalable, production-grade design



# Scientific Principles Behind Design

## 1. Domain-Aware Cleaning

- Physically implausible values (e.g., negative planetary radii) are clipped.
- Stellar metallicity (`st_met`) is imputed using **solar metallicity (0.0)** — a domain-standard reference.
- Missing masses and orbital periods are replaced using **robust statistics** (median), not mean, to avoid distortion by outliers.

**Why this matters:** In high-dimensional astrophysical space, precision is critical. A single NaN or outlier could skew a learning algorithm drastically.

## 2. Transformation Pipeline Construction

The preprocessing pipeline leverages `scikit-learn`'s modular composition:

- Logarithmic Transforms (`log1p`)  
Used on positively skewed variables such as `pl_rade`, `pl_insol`. Astrophysical magnitudes often span orders of magnitude, making log-scaling essential.
- Yeo-Johnson Power Transform  
Applied to variables like `st_met`, `pl_eqt`, which may contain both positive and negative values. This enables transformation to near-normality without data loss.
- Winsorization  
Instead of discarding outliers, which may hold astrophysical significance, this transformation caps them to a threshold. It balances robustness and sensitivity.
- StandardScaler  
Ensures that all features, regardless of origin (planetary or stellar), are zero-centered and unit-variance scaled — a prerequisite for many ML models like SVMs and XGBoost.

## 3. Feature Grouping with ColumnTransformer

Each transformation group (log, power, remaining) is applied independently using a `ColumnTransformer`, which:

- Preserves interpretability via named transformers (`log_features`, `power_features`, etc.)

- Supports **parallel transformation pipelines**, vital for scaling to future missions with 10x more exoplanet features
- Allows for **introspectable** pipelines, useful for regulatory traceability and feature auditing

#### 4. NamedFunctionTransformer for Interpretability

`FunctionTransformer` is subclassed to maintain feature traceability through transformation stages, which is **crucial for model explainability** in scientific publications or space mission audits.

Fit and Transform

- Cleans the input data (`_clean_data`)
- Builds and fits the pipeline (`build_pipeline`)
- Applies transformation
- Reconstructs a new `DataFrame` with **corrected feature names**, retaining scientific traceability post-transformation

This method ensures:

- High-fidelity mapping from raw NASA catalog entries to learning-optimized vectors
- Retention of planetary identifiers (`pl_name`, `hostname`) for downstream analysis

## Output

- A cleaned and transformed `pandas.DataFrame` (`processed_df`) ready for dimensionality reduction (e.g., PCA), classification (e.g., SVM), or regression (e.g., mass prediction).
- All preprocessing steps are **deterministic, modular, and auditable** — ideal for scientific reproducibility.

## Phase 4: Final Data Preparation and Cross-Validation Strategy

completes the scientific preprocessing pipeline by handling:

- High-fidelity missing data imputation
- Supervised learning target split
- Stratified cross-validation planning

We deploy a **10-fold Stratified K-Fold CV** for robust generalization assessment:

## Phase 5: Model Training, Evaluation & Superiority of XGBoost

`XGBoostTrainer` is a high-performance gradient boosting model using the [XGBoost](#) engine — designed to handle tabular data with nonlinear relationships, missing values, and imbalanced classes effectively.

`scale_pos_weight=1` reflects a balanced label distribution, but this parameter remains tunable based on class skew.

`calibrator="sigmoid"` activates probability calibration via Platt scaling, ensuring well-calibrated confidence scores for metrics like AUC and Brier loss.

`cv=5` enables internal cross-validation within the model training loop for optimal hyperparameter tuning or early stopping.




KNN, a distance-based non-parametric classifier, was tested but ultimately **excluded from production** due to **inferior generalization performance** on the habitable planet detection task.

The **ModelEvaluator** abstraction handles all evaluation loops across the cross-validation splits defined earlier.

It computes metrics like **accuracy**, **precision**, **recall**, **F1**, **AUROC**, and **AUPRC** per fold, then averages them for robust comparison.

# Why XGBoost Outperformed KNN (Decisively)

In your experiments, **XGBoost significantly and consistently outperformed KNN**, and here's why—rooted in statistical and computational principles:

 Aspect	 XGBoost	 KNN
Handling Feature Interactions	Learns nonlinear combinations via gradient trees	No learned interactions; purely distance-based
Scalability	Highly optimized, GPU-accelerated	Slows dramatically with large datasets
Noise Sensitivity	Robust via regularization ( <code>lambda</code> , <code>gamma</code> )	Extremely sensitive to outliers and irrelevant features
Missing Data	Native support via tree splitting logic	Requires explicit imputation and scaling
Calibration	Supports sigmoid calibration	Outputs raw discrete votes (poor probabilistic interpretation)
Class Imbalance	<code>scale_pos_weight</code> , early stopping, and weighted loss	No built-in mechanisms; requires resampling
Interpretability	Feature importances, SHAP values	Black-box distances; hard to explain

