# CODE DESCRIPTION

PART 1 – DATA PREPROCESSING

---

## 1. Uploading the Dataset

```
from google.colab import files
uploaded = files.upload()
```

**Explanation:**

- from google.colab import files imports the file-handling utilities provided by Google Colab.
- files.upload() opens a dialog box allowing the user to upload files from the local system.
- The uploaded file is stored temporarily in the Colab environment.

---

## 2. Verifying Uploaded Files

```
import os
os.listdir("/content")
```

**Explanation:**

- Import os imports the operating system module. - os.listdir("/content") lists all files in the Colab working directory.

---

## 3. Importing Pandas and Reading the Dataset

```
import pandas as pd

df = pd.read_csv(
    "/content/exponetial.csv",
    comment="#",
    low_memory=False
)

df.head()
```

**Explanation:**

- Pandas is imported for data manipulation.
- pd.read_csv() loads the CSV file into a DataFrame.
- comment="#" ignores commented lines in the file.
- low_memory=False ensures correct data type detection.

- df.head() displays the first five rows for inspection.

---

## 4. Installing Required Libraries

```
pip install pandas numpy matplotlib seaborn
```

**Explanation:**

- Installs required libraries for data analysis and visualization.
- This step is necessary in cloud environments like Google Colab.

---

## 5. Importing Libraries for Processing

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import zscore
```

**Explanation:**

- NumPy is used for numerical operations.
- Matplotlib and Seaborn are used for visualization.
- Z-score is imported for statistical analysis.

---

## 6. Reloading Dataset and Checking Row Count

```
df = pd.read_csv("exponetial.csv", comment="#", low_memory=False)
print("Original rows:", df.shape[0])
```

**Explanation:**

- Reloads the dataset to ensure a clean preprocessing start.
- df.shape[0] returns the number of rows.

---

## 7. Selecting Relevant Columns

```
df = df[[
    "pl_rade",
    "pl_bmasse",
    "pl_orbper",
    "pl_orbsmax",
```

```
    "pl_eqt",
    "pl_dens",
    "st_teff",
    "st_lum",
    "st_met",
    "st_spectype"
]]
```

**Explanation:**

- Retains only columns relevant to exoplanet habitability.
- Removes unnecessary attributes to reduce noise.

---

## 8. Renaming Columns

```
df.columns = [
    "planet_radius",
    "planet_mass",
    "orbital_period",
    "semi_major_axis",
    "equilibrium_temp",
    "planet_density",
    "star_temp",
    "star_luminosity",
    "star_metallicity",
    "star_type"
]
```

**Explanation:**

- Converts technical column names into readable and meaningful names.

---

## 9. Handling Missing Values

```
num_cols = df.select_dtypes(include=np.number).columns
df[num_cols] = df[num_cols].fillna(df[num_cols].median())
df["star_type"] = df["star_type"].fillna(df["star_type"].mode()[0])
```

**Explanation:**

- Numerical missing values are filled using the median.
- Categorical missing values are filled using the mode.

---

## 10. Removing Invalid Values

```
df = df[
   (df["planet_radius"] > 0) &
   (df["planet_mass"] > 0) &
   (df["equilibrium_temp"] > 0)
]
```

**Explanation:**

- Removes physically impossible values such as negative radius or temperature.

---

## 11. Outlier Handling Using IQR

```
def cap_iqr(data, col):
   Q1 = data[col].quantile(0.25)
   Q3 = data[col].quantile(0.75)
   IQR = Q3 - Q1
   lower = Q1 - 1.5 * IQR
   upper = Q3 + 1.5 * IQR
   data[col] = data[col].clip(lower, upper)
```

**Explanation:**

- Uses the Interquartile Range method to cap extreme values.
- Prevents data loss while reducing outlier impact.

---

## 12. Applying Outlier Treatment

```
for col in num_cols:
   cap_iqr(df, col)
```

**Explanation:**

- Applies IQR capping to all numerical columns.

---

## 13. Habitability Score Calculation

```
df["habitability_score"] = (
  (1 - abs(df["equilibrium_temp"] - 288) / 288).clip(0, 1) * 0.4 +
   (1 - abs(df["planet_radius"] - 1)).clip(0, 1) * 0.3 +
   (1 / (1 + abs(df["semi_major_axis"] - 1))).clip(0, 1) * 0.3
)
```

**Explanation:**

- Calculates a weighted habitability score based on Earth similarity.

- Temperature contributes 40%, radius 30%, and orbital distance 30%.

---

## 14. Feature Engineering (Encoding Star Type)

```
df["star_class"] = df["star_type"].astype(str).str[0]
df = pd.get_dummies(df, columns=["star_class"], drop_first=True)
```

**Explanation:**

- Extracts stellar spectral class.
- Converts categorical values into numeric features.

---

## 15. Creating Binary Habitability Label

```
df["habitable"] = (df["habitability_score"] >= 0.4).astype(int)
```

**Explanation:**

- Creates a binary classification label.
- 1 indicates habitable, 0 indicates non-habitable.

---

## 16. Removing Duplicates and Saving File

```
df = df.drop_duplicates()
df.to_csv("preprocessed.csv", index=False)
```

**Explanation:**

- Removes duplicate records.
- Saves the cleaned dataset.

---

## 17. Verifying Processed Dataset

```
df = pd.read_csv("/content/preprocessed.csv")
df.head()
```

**Explanation:** -

- Reloads the processed file for verification.

---

# PART 2 – ML MODEL TRAINING

## 1. Importing Required Libraries

```
import pandas as pd
import numpy as np
import joblib
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
classification_report, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

**Explanation:**

- Pandas and NumPy are used for data handling and numerical operations.
- Joblib is used to save trained machine learning models.
- Matplotlib is used for visualization.
- Scikit-learn modules are imported for splitting data, scaling features, building pipelines, and evaluating models.
- Logistic Regression, Random Forest, and XGBoost are imported as candidate models.

---

## 2. Loading the Preprocessed Dataset

```
df = pd.read_csv("preprocessed.csv")
```

**Explanation:**

- Loads the cleaned and feature-engineered dataset produced during preprocessing.
- This dataset contains numerical features and a target column named `habitable`.

---

## 3. Separating Features and Target

```
X = df.drop("habitable", axis=1)
y = df["habitable"]
```

**Explanation:**

- `X` contains all independent features used for prediction.
- `y` is the target variable representing habitability class (1 = habitable, 0 = non-habitable).

## 4. Train–Test Split

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

**Explanation:**

- Splits data into 80% training and 20% testing sets.
- `random_state=42` ensures reproducibility.
- `stratify=y` maintains class balance in both sets.

---

## 5. Building a Machine Learning Pipeline

```
pipeline_rf = Pipeline([
    ("scaler", StandardScaler()),
    ("model", RandomForestClassifier(random_state=42))
])
```

**Explanation:**

- A pipeline is created to combine preprocessing and model training.
- StandardScaler normalizes feature values.
- Random Forest is used as the classifier.
- Pipelines prevent data leakage and improve reproducibility.

---

## 6. Training the Model

```
pipeline_rf.fit(X_train, y_train)
```

**Explanation:**

- Trains the Random Forest model using the training dataset.
- The scaler is fitted only on training data.

---

## 7. Making Predictions

```
y_pred = pipeline_rf.predict(X_test)
y_prob = pipeline_rf.predict_proba(X_test)[:, 1]
```

**Explanation:**

- `predict()` gives class predictions (0 or 1).
- `predict_proba()` gives probability of the positive class (habitable).

## 8. Evaluating Model Performance

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_prob)
```

**Explanation:**

- Accuracy measures overall correctness.
- Precision measures reliability of positive predictions.
- Recall measures ability to detect habitable planets.
- F1-score balances precision and recall.
- ROC-AUC measures probability-based class discrimination.

---

## 9. Printing Evaluation Reports

```
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

**Explanation:**

- Classification report provides precision, recall, and F1-score per class.
- Confusion matrix shows true vs predicted classifications.

---

## 10. Saving the Trained Model

```
joblib.dump(pipeline_rf, "models/random_forest.pkl")
```

**Explanation:**

- Saves the trained model for reuse without retraining.
- Ensures deployment readiness.

---

## 11. FINAL MODEL PERFORMANCE SUMMARY

MODEL: Random Forest Classifier

Accuracy : 99%
Precision : 98%
Recall : 98%
F1-Score : 98%
ROC-AUC : 99%

---

## Final Conclusion

This machine learning pipeline successfully trains, evaluates, and saves a predictive model for exoplanet habitability. The use of pipelines, multiple evaluation metrics, and probability-based scoring ensures robust, interpretable, and scientifically meaningful predictions.