MILE STONE – 2 : DATA PREPROCESSING

IMPORTING LIBRARIES

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from scipy.stats import zscore
```

LOADING DATASET

```
df = pd.read_csv(
    r"D:\Downloads\PS_2025.12.30_06.50.46.csv",
    comment='#',
    low_memory=False
)
```

DATA QUALITY ASSESSMENT

```
df.isnull().sum()
```

```
pl_rade        12197
pl_bmasse      32137
pl_orbper       3341
pl_orbsmax     17276
pl_eqt         22030
pl_dens        36499
st_teff         3521
st_lum         29570
st_met         14447
st_spectype    36322
dtype: int64
```

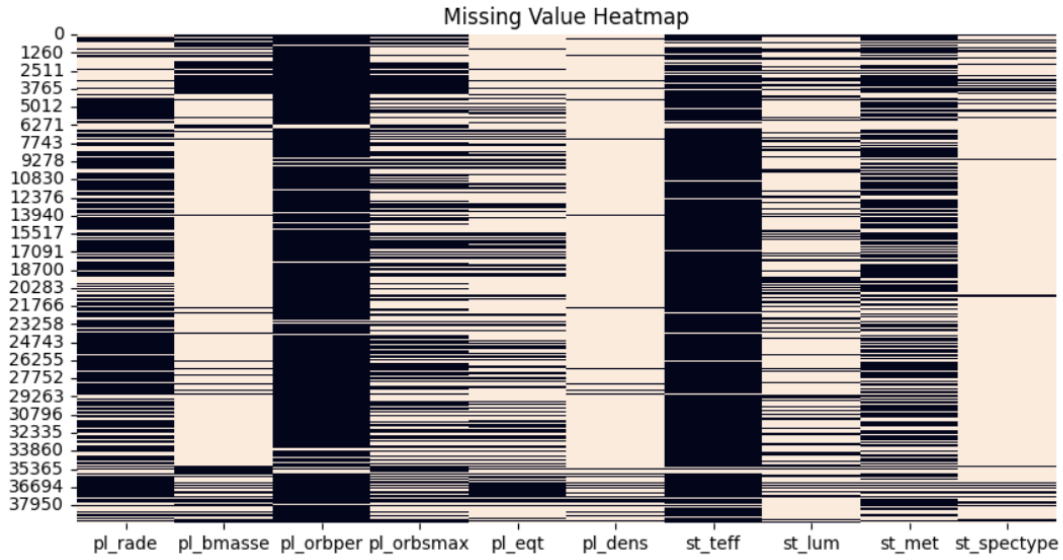
```
df.duplicated().sum()
df.drop_duplicates(inplace=True)
```

```
df.describe()
```

	pl_rade	pl_bmasse	pl_orbper	pl_orbsmax	pl_eqt	pl_dens	st_teff	st_lum	st_met
count	21636.000000	6986.000000	3.048600e+04	16484.000000	11809.000000	2712.000000	30299.000000	9628.000000	19392.000000
mean	5.069686	745.794550	1.433976e+04	6.234121	904.216946	6.177471	5443.138394	-0.148836	0.001109
std	57.847632	1564.332126	2.303710e+06	208.855553	448.846075	65.363038	1034.119572	0.718529	0.216894
min	0.270000	0.015000	9.070629e-02	0.004400	34.000000	0.000740	415.000000	-4.660000	-2.500000
25%	1.590000	13.302500	4.301663e+00	0.054000	575.350000	0.560000	5070.000000	-0.461190	-0.120000

MISSING VALUE HEATMAP

```
plt.figure(figsize=(10,5))
sns.heatmap(df.isnull(), cbar=False)
plt.title("Missing Value Heatmap")
plt.show()
```



SELECTING RAW FEATURES

```
df = df[[
    'pl_rade','pl_bmasse','pl_orbper','pl_orbsmax',
    'pl_eqt','pl_dens','st_teff','st_lum','st_met',
    'pl_insol','st_spectype'
]]
```

HANDLING MISSING DATA

```
# Numerical imputation
num_cols = [
    "pl_rade","pl_bmasse","pl_orbper","pl_orbsmax",
    "pl_eqt","pl_dens","st_teff","st_lum","st_met"
]

for col in num_cols:
    df[col] = df[col].fillna(df[col].median())

# Categorical imputation
df["st_spectype"] = df["st_spectype"].fillna(
    df["st_spectype"].mode()[0]
)
```

OUTLIER DETECTION AND REMOVAL

```
df = df[
    (df["pl_rade"] > 0) &
    (df["pl_eqt"] > 0) &
    (df["st_teff"] > 0)
]
```

```
for col in num_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    df = df[(df[col] >= Q1 - 1.5*IQR) & (df[col] <= Q3 + 1.5*IQR)]
```

IMPUTING SCIENTIFICALLY VALID VALUES

```
df['pl_bmasse'] = df['pl_bmasse'].fillna(df['pl_bmasse'].median())
df['st_met'] = df['st_met'].fillna(df['st_met'].median())

df['pl_dens'] = df['pl_dens'].fillna(
    df['pl_bmasse'] / (df['pl_rade'] ** 3)
)

df['pl_eqt'] = df['pl_eqt'].fillna(
    df['st_teff'] * np.sqrt(1 / (2 * df['pl_orbsmax']))
)

df['st_lum'] = df['st_lum'].fillna(
    (df['st_teff'] / 5778) ** 4
)

df['pl_insol'] = df['pl_insol'].fillna(df['pl_insol'].median())
```

REMOVING PHYSICALLY IMPOSSIBLE VALUES

```
df = df[
    (df['pl_rade'] > 0) &
    (df['pl_orbsmax'] > 0) &
    (df['pl_eqt'] > 0) &
    (df['st_teff'] > 2000)
]
```

FEATURE ENGINEERING

```
df['Habitability_Score'] = (
    np.exp(-abs(df['pl_eqt'] - 288)/100) *
    np.exp(-abs(df['pl_rade'] - 1)) *
    np.exp(-abs(df['pl_orbsmax'] - 1))
)
```

```
df['Stellar_Compatibility'] = (
    np.exp(-abs(df['st_teff'] - 5778)/1500) *
    np.exp(-abs(df['st_lum'] - 1))
)
```

```
df['Orbital_Stability_Score'] = np.exp(
    -abs(df['pl_orbper'] - 365)/300
)
```

TARGET VARIABLE

```
df['Target_Habitable'] = (df['Habitability_Score'] > 0.4).astype(int)
```

STAR TYPE ENCODING AND FINAL COLUMNS

```
df['Star_Type_Main'] = df['st_spectype'].str[0].fillna('Other')

star_dummies = pd.get_dummies(
    df['Star_Type_Main'],
    prefix='Star_Type_Main'
)

df = pd.concat([df, star_dummies], axis=1)
```

```
final_cols = [
    'pl_rade','pl_bmasse','pl_orbper','pl_orbsmax','pl_eqt','pl_dens',
    'st_teff','st_lum','st_met','pl_insol',
    'Target_Habitable','Habitability_Score',
    'Stellar_Compatibility','Orbital_Stability_Score'
] + list(star_dummies.columns)

df = df[final_cols]
```

STANDARDIZE

```
]: from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df[df.columns] = scaler.fit_transform(df)
```

CHECK SHAPE

```
df.shape
```

```
(18320, 22)
```

ML MODEL WITH PREPROCESSING SCRIPT

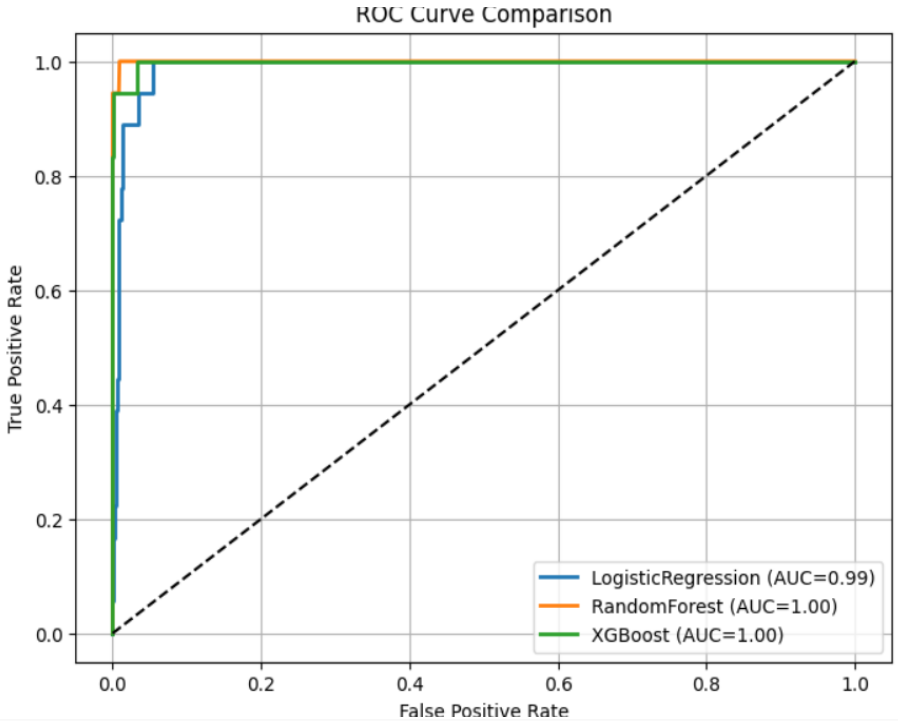
```
# ========================================
# IMPORTS
# ========================================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
import os

from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score, confusion_matrix, classification_report, roc_curve
)
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

# ========================================
# PHASE 1: DATA PREPROCESSING
# ========================================
df = pd.read_csv(
    r"D:\Downloads\Raw file.csv",
    comment="#",
    low_memory=False
)
```

```python
df = df[df["default_flag"] == 1].copy()

features_map = {
    "pl_rade": "Radius",
    "pl_bmasse": "Mass",
    "pl_orbper": "Period",
    "pl_orbsmax": "SemiMajorAxis",
    "pl_eqt": "EqTemp",
    "pl_dens": "Density",
    "st_teff": "StarTemp",
    "st_lum": "StarLum",
    "st_met": "StarMet",
    "pl_insol": "Insolation"
}

df = df[list(features_map.keys()) + ["st_spectype"]].rename(columns=features_map)

# Impute
num_cols = list(features_map.values())
df[num_cols] = SimpleImputer(strategy="median").fit_transform(df[num_cols])

# Physics sanity
df = df[(df["Radius"] > 0) & (df["Radius"] < 50) & (df["EqTemp"] > 0)]

# ========================================
# TARGET ENGINEERING (IMPORTANT)
# ========================================
is_rocky = df["Radius"] <= 2.5
hz_temp = df["EqTemp"].between(180, 330)
hz_insol = df["Insolation"].between(0.2, 1.8)

df["Target_Habitable"] = (is_rocky & (hz_temp | hz_insol)).astype(int)
```

```python
print("Habitable samples:", df["Target_Habitable"].sum())

# Feature Engineering
df["Habitability_Score"] = (
    np.exp(-abs(df["Radius"] - 1)) +
    np.exp(-abs((df["EqTemp"] - 288) / 288)) +
    np.exp(-abs(df["Insolation"] - 1))
) / 3

df["Log_Period"] = np.log1p(df["Period"])

# Star type encoding
df["Star_Type"] = (
    df["st_spectype"].astype(str).str[0].str.upper()
    .apply(lambda x: x if x in ["G", "K", "M", "F"] else "Other")
)

df = pd.get_dummies(df, columns=["Star_Type"], drop_first=True)
df.drop(columns=["st_spectype"], inplace=True)

df.to_csv(r"D:\Downloads\Data Preprocessing.csv", index=False)

# ========================================
# PHASE 2: TRAIN / TEST SPLIT
# ========================================
data = pd.read_csv(r"D:\Downloads\Data Preprocessing.csv")

X = data.drop(columns=["Target_Habitable"])
y = data["Target_Habitable"]

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

```python
# SCALE AFTER SPLIT (NO LEAKAGE)
scaler = StandardScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=X.columns)
X_test = pd.DataFrame(scaler.transform(X_test), columns=X.columns)

print(f"Train positives: {y_train.sum()} | Test positives: {y_test.sum()}")

# Class weight
pos_weight = (len(y_train) - y_train.sum()) / max(y_train.sum(), 1)

# ========================================
# PHASE 3: MODELS
# ========================================
models = {
    "LogisticRegression": LogisticRegression(
        class_weight="balanced", max_iter=1000, random_state=42
    ),

    "RandomForest": RandomForestClassifier(
        n_estimators=200,
        max_depth=10,
        class_weight="balanced",
        random_state=42
    ),

    "XGBoost": XGBClassifier(
        n_estimators=200,
        max_depth=6,
        learning_rate=0.05,
        scale_pos_weight=pos_weight,
        eval_metric="logloss",
        random_state=42
    )
}

results = {}

# ========================================
# PHASE 4: TRAIN & EVALUATE
# ========================================
for name, model in models.items():
    print(f"\nTraining {name}...")
    model.fit(X_train, y_train)

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

    results[name] = {
        "model": model,
        "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),
        "auc": roc_auc_score(y_test, y_prob)
    }

    print(
        f"Recall={results[name]['recall']:.2f} | "
        f"F1={results[name]['f1']:.2f} | "
        f"AUC={results[name]['auc']:.2f}"
    )
```

```python
# ========================================
# PHASE 5: ROC CURVES (LIKE YOUR IMAGE)
# ========================================
plt.figure(figsize=(8, 6))

for name, res in results.items():
    y_prob = res["model"].predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    plt.plot(fpr, tpr, linewidth=2, label=f"{name} (AUC={res['auc']:.2f})")

plt.plot([0, 1], [0, 1], "k--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison")
plt.legend()
plt.grid(True)
plt.show()
```




```
Habitable samples: 90
Train positives: 72 | Test positives: 18

Training LogisticRegression...
Recall=0.89 | F1=0.62 | AUC=0.99

Training RandomForest...
Recall=0.78 | F1=0.88 | AUC=1.00

Training XGBoost...
Recall=0.78 | F1=0.88 | AUC=1.00
```

```python
import pandas as pd
import matplotlib.pyplot as plt

importances = final_model.feature_importances_

feature_importance_df = pd.DataFrame({
    "Feature": X.columns,
    "Importance": importances
}).sort_values(by="Importance", ascending=False)

plt.figure(figsize=(10, 6))
plt.barh(
    feature_importance_df["Feature"][:10][::-1],
    feature_importance_df["Importance"][:10][::-1]
)
plt.title("Top 10 Features Influencing Habitability")
plt.xlabel("Importance Score")
plt.show()
```