1. Overview: Data Preparation and Feature Engineering

In our web-based wildlife conservation awareness project—aligned with the UN Sustainable Development Goals (SDG 15: Life on Land)—the data preparation and feature engineering phase plays a vital role in building a reliable and effective machine learning system.

This phase ensures that the raw data collected from diverse sources (such as Kaggle and the IUCN Red List) is cleaned, structured, and transformed into meaningful inputs that can be understood by the model. Wildlife-related datasets often include inconsistencies, missing values, and varied formats due to differences in data collection methods. Therefore, we focused on tasks like:

- Cleaning and integrating species information across datasets

- Addressing missing values and standardizing categorical variables (e.g., threat types, habitats)

- Engineering informative features such as threat severity scores

and conservation priority indices

- Scaling and encoding data for optimal model performance

These efforts directly support our goal of raising awareness about endangered species and their threats, enabling our AI models to provide accurate, interpretable insights that can inform conservation strategies and engage the public.

2. Data Collection

For our wildlife conservation awareness project, we collected data from two primary sources:

1. Kaggle Datasets

We used datasets related to wildlife conservation, endangered species, and ecological threats. A key dataset was the [Wildlife Conservation dataset], which includes details such as species names, conservation status, population trends, and threat types.

2. IUCN Red List (International Union for Conservation of Nature)

The IUCN Red List provided structured data on the global conservation status of species. This included information like:

- Species scientific/common names

- Red List category (e.g., Vulnerable, Endangered)

- Habitat and geographic distribution

- Threat levels and trends

-Preprocessing Steps During Data Collection:

- Format Normalization:

Datasets from different sources came in CSV and JSON formats. We converted all data into Pandas DataFrames for consistency.

- Column Unification:

Harmonized column names across datasets (e.g., `species\_name`, `status`, `habitat`) to enable easier merging and analysis.

- Data Filtering:

Filtered out incomplete entries and focused on species with defined threat levels and recent assessments.

- Merging Datasets:

Used species names and IUCN IDs to merge Kaggle and Red List data, ensuring a rich, unified dataset combining ecological, geographical, and conservation-related attributes.

- Initial Data Checks:

Verified data types, removed duplicates, and checked for null values to ensure integrity before deeper processing.

This step laid the foundation for the next phases, allowing us to build a robust dataset for awareness-driven machine learning modeling.

3. Data Cleaning

Cleaning the raw data was essential to ensure the accuracy and reliability of our machine learning models in the wildlife conservation awareness project. Due to the nature of data sourced from Kaggle and the IUCN Red List, several quality issues were identified and addressed:

1. Handling Missing Values

- Population Size & Range Data:

- Missing values in numerical fields like population size or range area were imputed using the median to avoid skewing the data with extreme values.

- Categorical Fields (e.g., Habitat Type, Threats):

- Missing categorical values were filled with `"Unknown"` or `"Not Reported"` to retain these records without introducing false information.

- Red List Category or Assessment Year:

- Entries missing critical conservation labels were excluded from training but retained for awareness visualization.

2. Removing Duplicates

- Duplicate records (especially species appearing multiple times with different formats) were identified using `species\_name` and `iucn\_id` and removed or consolidated.

3. Outlier Detection and Treatment

-Numerical Features (e.g., population size, range area):

- Outliers were detected using box plots and Z-score thresholds.

- Extreme outliers were either:

- Capped at the 95th percentile, or

- Removed if biologically implausible (e.g., a single mammal with population size of 10 million).

4. Standardization of Categorical Data

- Values like `"Forest"`, `"forest"`, and `"FOREST"` were standardized to lowercase: `"forest"`.

- Multi-value fields (e.g., multiple threats listed in one field) were split into lists for easier encoding later.

5. Consistency Checks

- Ensured species names were consistently formatted (`Genus species`) and trimmed whitespace.

- Verified that all entries had valid ISO country codes or region identifiers when used for geolocation.

By carefully cleaning the data, we ensured that the downstream processes—EDA, feature engineering, and model training—were built on a solid, accurate foundation, minimizing noise and improving predictive performance.

4. Exploratory Data Analysis (EDA)

We analyzed the dataset to better understand species status, threats, and patterns.

Key Findings:

-Conservation Status: Most species are Least Concern, but many are Endangered or Critically Endangered.

-Habitats: Forests and wetlands are the most common habitats for threatened species.

-Top Threats: Habitat loss, hunting, and climate change are the main threats.

-Population Trends: Many endangered species have a declining population.

-Correlations: Weak link between population size and threat level — used this to build new features later.

5. Feature Engineering

In this phase, we enhanced the dataset by creating new features and transforming existing ones to improve model performance and better represent conservation-related patterns.

New Features Created

- Threat Score:

A numerical score was generated by assigning values to each identified threat (e.g., habitat loss, hunting, climate change) and summing them.

Rationale: To quantify the overall level of risk a species faces, enabling the model to distinguish between low-risk and high-risk species.

- Conservation Priority Score:

A custom index calculated using conservation status, population trend, and ecological importance.

Rationale: To help prioritize species that require immediate conservation efforts.

- Habitat Count:

Derived by counting the number of unique habitats listed for each species.

Rationale: Species with fewer habitats may be more vulnerable to habitat destruction or environmental changes.

Transformation of Existing Features:

- Population Size (Log Transformation):

Applied logarithmic scaling to reduce the impact of extreme values.

Rationale: To normalize highly skewed data, ensuring the model isn't biased by very large population numbers.

These feature engineering steps were essential in transforming raw ecological data into meaningful inputs that improved the predictive accuracy and interpretability of our machine learning model.

6. Data Transformation

In this step, we made sure the data was in the right format and scale for the machine learning model. Here’s what we did:

1. Scaling and Normalizing the Data:

- Min-Max Scaling:

Some numbers were much larger than others. We used Min-Max scaling to make sure all numbers are between 0 and 1, which helps the model learn better.

Code:

```python

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df[['conservation\_priority']] = scaler.fit\_transform(df[['conservation\_priority']])

2. Encoding Categorical Data:

- One-Hot Encoding for Habitat Type:

The Habitat Type feature had categories like Forest and Savanna. We turned each of these categories into a separate column (like “Is it a Forest?”), so the model can understand them.

Code:

```python

df = pd.get\_dummies(df, columns=['habitat\_type'], drop\_first=True)

- Label Encoding for Conservation Status:

Conservation Status was turned into numbers (e.g., Least Concern = 0, Endangered = 3). This helps the model handle it better.

Code:

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

df['conservation\_status'] = label\_encoder.fit\_transform(df['conservation\_status'])

These transformations helped make the data easier for the machine learning model to understand and work with.

Model Exploration

1. Model Selection:

We need to choose a model that can predict how at risk a species is. Some common models include Decision Trees, Random Forests, and Logistic Regression. These models work well because they can handle both numerical and categorical data.

2.Training the Model:

To train the model, we followed these steps:

Splitting the Data:

We split the dataset into two parts: one for training the model and one for testing it. Typically, we use 80% of the data for training and 20% for testing.

code:

from sklearn.model\_selection import train\_test\_split

X = df.drop('target', axis=1) # Features (everything except the target variable)

y = df['target'] # Target variable (species risk level or conservation status)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

8. Model Evaluation

After training the model, we evaluated its performance using various metrics to ensure it was making accurate predictions. Here are the key metrics and visualizations used for this assessment:

1. Accuracy:

Accuracy is the percentage of correct predictions made by the model. It is the most straightforward metric but can be misleading if the dataset is imbalanced (e.g., if one class is much larger than the other).

Code:

from sklearn.metrics import accuracy\_score

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy: ", accuracy)

2. Confusion Matrix:

A confusion matrix shows how well the model is performing by comparing predicted labels with actual labels. It displays:

True Positives (TP): Correctly predicted positive outcomes.

True Negatives (TN): Correctly predicted negative outcomes.

False Positives (FP): Incorrectly predicted as positive.

False Negatives (FN): Incorrectly predicted as negative.

This matrix helps us understand if the model is favoring one class over another.

code:

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

cm = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])

plt.title('Confusion Matrix')

plt.show()

3. ROC Curve and AUC (Area Under the Curve):

The ROC curve shows how well the model distinguishes between classes by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at various thresholds. The AUC (Area Under the Curve) represents the overall ability of the model to discriminate between classes — the higher the AUC, the better the model.

Code:

from sklearn.metrics import roc\_curve, auc

fpr, tpr, \_ = roc\_curve(y\_test, model.predict\_proba(X\_test)[:,1])

roc\_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='blue', label=f'ROC curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic Curve')

plt.legend(loc='lower right')

plt.show()

4. Precision, Recall, and F1-Score:

These metrics give more detailed information about the model’s performance, especially when dealing with imbalanced datasets.

Precision: How many of the predicted positives are actually correct.

Recall: How many of the actual positives were correctly predicted.

F1-Score: The harmonic mean of precision and recall, balancing both metrics.

code:

from sklearn.metrics import precision\_score, recall\_score, f1\_score

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

print(f'Precision: {precision}')

print(f'Recall: {recall}')

print(f'F1-Score: {f1}')

### \*\*9. Code Implementation\*\*

Below are the key code snippets used for Data Preparation/Feature Engineering and Model Exploration. Each section is commented to explain what each part of the code is doing.

1. Data Preparation & Feature Engineering

Code:

# Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler, LabelEncoder

# Load the dataset

df = pd.read\_csv('species\_data.csv')

# Feature Engineering: Log transformation of population size to reduce skewness

df['log\_population'] = np.log(df['population\_size'] + 1) # Added 1 to avoid log(0)

# Feature Engineering: Create a threat score by summing individual threat types

df['threat\_score'] = df[['habitat\_loss', 'hunting', 'climate\_change']].sum(axis=1)

# Feature Engineering: Create a conservation priority score (higher value means more priority)

df['conservation\_priority'] = df['conservation\_status'] \* df['population\_trend']

# Feature Engineering: Count the number of habitats for each species

df['habitat\_count'] = df['habitats'].apply(lambda x: len(x.split(',')))

# Scaling: Min-Max normalization for the conservation priority score

scaler = MinMaxScaler()

df[['conservation\_priority']] = scaler.fit\_transform(df[['conservation\_priority']])

# Encoding: Label Encoding for conservation status (Least Concern = 0, Endangered = 3)

label\_encoder = LabelEncoder()

df['conservation\_status'] = label\_encoder.fit\_transform(df['conservation\_status'])

# Encoding: One-Hot Encoding for habitat types (e.g., Forest, Wetland)

df = pd.get\_dummies(df, columns=['habitat\_type'], drop\_first=True)

# Check the data after transformations

print(df.head())

2. Model Training

# Import necessary libraries for model training

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.metrics import accuracy\_score, confusion\_matrix, roc\_curve, auc

# Split the data into features (X) and target (y)

X = df.drop('target', axis=1) # Drop target column for features

y = df['target'] # Target column (species risk level)

# Split the data into training and test sets (80% for training, 20% for testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize the RandomForest model with 100 decision trees

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Train the model on the training data

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print("Model Accuracy: ", accuracy)

# Cross-validation to assess the model's stability

cv\_scores = cross\_val\_score(model, X, y, cv=5)

print("Average 5-Fold CV Score: ", cv\_scores.mean())

3. Model Evaluation

# Confusion Matrix to understand prediction results

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix: \n", cm)

# Plot confusion matrix

import seaborn as sns

import matplotlib.pyplot as plt

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])

plt.title('Confusion Matrix')

plt.show()

# ROC Curve to evaluate model's performance

fpr, tpr, \_ = roc\_curve(y\_test, model.predict\_proba(X\_test)[:,1])

roc\_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='blue', label=f'ROC curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic Curve')

plt.legend(loc='lower right')

plt.show()

# Precision, Recall, and F1-Score

from sklearn.metrics import precision\_score, recall\_score, f1\_score

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

print(f'Precision: {precision}')

print(f'Recall: {recall}')

print(f'F1-Score: {f1}')

-Explanation of Key Sections:

1.Data Preparation & Feature Engineering:

- Log Transformation: Applied to the population size to make the data less skewed.

- Threat Score: A sum of all the threats a species faces, which helps the model identify overall danger.

- Conservation Priority: A custom score based on the species’ conservation status and population trend.

- Min-Max Scaling: Scaled the conservation priority score to be between 0 and 1.

- Label and One-Hot Encoding: These steps convert categorical data (like habitat type and conservation status) into a numerical format.

2. Model Training:

- We split the data into training and test sets to evaluate the model’s performance on unseen data.

- A Random Forest model was used because it's effective at handling both numerical and categorical data.

3. Model Evaluation:

- Accuracy: Measures the overall correctness of the model.

- Confusion Matrix: Shows how many times the model correctly predicted each class and how many times it made mistakes.

- ROC Curve & AUC: Helps evaluate how well the model distinguishes between classes.

- Precision, Recall, F1-Score: Provide more detailed metrics, especially in cases of imbalanced classes.