

---

 User:

crocodile dataset

You are my senior data science coach. Guide me step by step through Assignment 1: using the `crocodile_dataset.csv` dataset to build a predictive model with the CRISP-DM framework. Rules: Break the project into small chunks (Business Understanding → Data Understanding → Data Preparation → Modeling → Evaluation → Insights). At each step: Explain the goal in simple words Give me Colab-ready Python code Tell me what result to screenshot for my report After each chunk, stop and say: "✅ Your turn: run this and take a screenshot. Then type CONTINUE to get the next step." Keep explanations very short and beginner-friendly, like teaching a kid. Use sklearn, pandas, and matplotlib only. The dataset is `crocodile_dataset.csv`. Target = "Conservation Status". Metrics = Accuracy and F1.

---

ChatGPT:

## Step 1 — Business Understanding

**Goal (kid-simple):** We want a tiny robot to guess each crocodile's **Conservation Status** from its features. We'll judge the robot by **Accuracy** and **F1** (how often it's right, and how well it finds each class).

**What we'll do now:** set the project knobs (target, metrics, random seed) and print a short plan so your notebook starts clean.

### Colab-ready code

```
python
# === Step 1: Business Understanding ===
# Keep this cell at the top of your notebook.

import pandas as pd
import matplotlib.pyplot as plt
from sklearn import __version__ as sklearn_version

# Project settings
RANDOM_STATE = 42
DATA_PATH = "crocodile_dataset.csv"  # put crocodile_dataset.csv in the same folder as the notebook
TARGET = "Conservation Status"
METRICS = ["accuracy", "f1"]

# Friendly intro printout
print("Assignment 1 – CRISP-DM")
print("Step: Business Understanding")
print("-----")
print("Goal: Predict each crocodile's Conservation Status.")
print(f"Target column: {TARGET}")
print(f"Metrics: {' '.join(m.upper() for m in METRICS)}")
print(f"Tools: pandas, scikit-learn {sklearn_version}, matplotlib")
print(f>Data file expected at: {DATA_PATH}")
print(f"Random seed: {RANDOM_STATE}")

# (Optional) Tiny visual header so you have something nice to screenshot
plt.figure(figsize=(6, 1.5))
plt.axis("off")
plt.text(0.5, 0.5, "CRISP-DM • Business Understanding", ha="center", va="center", fontsize=14)
plt.show()
```

### What to screenshot

- The printed block (Goal/Target/Metrics/Tools) **and** the small Matplotlib banner.

✓ Your turn: run this and take a screenshot. Then type CONTINUE to get the next step.

User:

Continue

ChatGPT:

## Step 2 — Data Understanding

**Goal (kid-simple):** Peek at the data. Count rows/columns, spot missing stuff, see what the target looks like.

### Colab-ready code

```
python
# === Step 2: Data Understanding ===
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

DATA_PATH = "crocodile_dataset.csv" # same as Step 1
TARGET = "Conservation Status"

# 1) Load
df = pd.read_csv(DATA_PATH)
assert TARGET in df.columns, f"Target '{TARGET}' not found. Found: {list(df.columns)}"

# 2) Basic shape/info
n_rows, n_cols = df.shape
print(f"Rows: {n_rows}, Columns: {n_cols}")
print("\nColumn dtypes:")
print(df.dtypes)

# Show first 5 rows
display(df.head())

# 3) Missing values
missing_counts = df.isna().sum().sort_values(ascending=False)
missing_pct = (missing_counts / len(df)).round(3)
missing_table = pd.DataFrame({"missing_count": missing_counts, "missing_pct": missing_pct})
display(missing_table[missing_table["missing_count"] > 0])

# 4) Identify numeric vs categorical (exclude target from features bucket)
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
cat_cols = [c for c in df.columns if c not in num_cols]
if TARGET in num_cols:
    num_cols.remove(TARGET)
if TARGET in cat_cols:
    cat_cols.remove(TARGET)

print("\nNumeric feature columns:", num_cols)
print("Categorical feature columns:", cat_cols)

# Quick stats for numeric
if num_cols:
    display(df[num_cols].describe().T)

# 5) Target distribution
target_counts = df[TARGET].value_counts(dropna=False)
target_props = (target_counts / target_counts.sum()).round(3)
print("\nTarget counts:")
print(target_counts)
print("\nTarget proportions:")
print(target_props)

plt.figure(figsize=(6,4))
bars = plt.bar(target_counts.index.astype(str), target_counts.values)
plt.title("Conservation Status – Class Balance")
plt.xlabel("Class")
plt.ylabel("Count")
# annotate percentages on bars
for bar, p in zip(bars, target_props.values):
```

```
plt.text(bar.get_x() + bar.get_width()/2, bar.get_height(), f"{p*100:.1f}%",
         ha='center', va='bottom', fontsize=9)
plt.tight_layout()
plt.show()

# 6) Quick look at numeric shapes (up to 6 histograms)
to_plot = num_cols[:6]
if to_plot:
    fig, axes = plt.subplots(nrows=int(np.ceil(len(to_plot)/3)), ncols=3, figsize=(10, 6))
    axes = axes.flatten()
    for ax, col in zip(axes, to_plot):
        ax.hist(df[col].dropna(), bins=20)
        ax.set_title(col)
        ax.set_xlabel("value"); ax.set_ylabel("freq")
    # hide any extra axes
    for j in range(len(to_plot), len(axes)):
        fig.delaxes(axes[j])
    fig.suptitle("Numeric Feature Distributions (first 6)", y=1.02)
    plt.tight_layout()
    plt.show()

# 7) Correlation heatmap (numeric-only, if at least 2)
if len(num_cols) >= 2:
    corr = df[num_cols].corr(numeric_only=True)
    plt.figure(figsize=(6,5))
    im = plt.imshow(corr, aspect='auto')
    plt.colorbar(im, fraction=0.046, pad=0.04)
    plt.xticks(ticks=range(len(num_cols)), labels=num_cols, rotation=90)
    plt.yticks(ticks=range(len(num_cols)), labels=num_cols)
    plt.title("Feature Correlation (numeric)")
    plt.tight_layout()
    plt.show()
```

### What to screenshot

- The dataframe **head()** + the printed shape/dtypes.
- The **missing values table** (if any).
- The **Conservation Status bar chart**.
- (Nice-to-have) The **numeric histograms** or **correlation heatmap**.

✓ **Your turn: run this and take a screenshot. Then type CONTINUE to get the next step.**

-----  
User:

Continue

-----  
ChatGPT:

## Step 3 — Data Preparation

**Goal (kid-simple):** Clean the columns, split into train/test, and build a **preprocessing pipeline** (fix missing values, encode text, scale numbers).

### Colab-ready code

```
python
# === Step 3: Data Preparation ===
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler

# Settings (keep consistent across steps)
DATA_PATH = "crocodile_dataset.csv"
TARGET = "Conservation Status"
RANDOM_STATE = 42
```

```

# 1) Load
df = pd.read_csv(DATA_PATH)
assert TARGET in df.columns, f"Target '{TARGET}' not found."

# 2) Separate features/target
X = df.drop(columns=[TARGET])
y = df[TARGET].astype(str) # strings are fine for classifiers

# 3) (Tiny cleanup) Drop constant columns in X (no info)
const_cols = [c for c in X.columns if X[c].nunique(dropna=True) <= 1]
if const_cols:
    print("Dropping constant columns:", const_cols)
    X = X.drop(columns=const_cols)

# 4) Train/Test split (stratify = keep class balance)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=RANDOM_STATE
)

# 5) Identify numeric/categorical columns
num_cols = X_train.select_dtypes(include="number").columns.tolist()
cat_cols = [c for c in X_train.columns if c not in num_cols]

print("Numeric cols:", num_cols)
print("Categorical cols:", cat_cols)

# 6) Build preprocessing:
# - Numeric: median impute -> standardize
# - Categorical: most_frequent impute -> one-hot encode
num_pipe = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

cat_pipe = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore", sparse=False))
])

transformers = []
if num_cols: transformers.append(("num", num_pipe, num_cols))
if cat_cols: transformers.append(("cat", cat_pipe, cat_cols))

preprocessor = ColumnTransformer(transformers=transformers, remainder="drop")

# 7) Fit on training and transform both splits
X_train_proc = preprocessor.fit_transform(X_train)
X_test_proc = preprocessor.transform(X_test)

# 8) Human-friendly feature names
feature_names = []
if num_cols:
    feature_names += num_cols
if cat_cols:
    ohe = preprocessor.named_transformers_["cat"].named_steps["onehot"]
    feature_names += list(ohe.get_feature_names_out(cat_cols))

# 9) Quick reports
print("\nShapes after preprocessing:")
print("X_train_proc:", X_train_proc.shape, " | X_test_proc:", X_test_proc.shape)
print("y_train:", y_train.shape, " | y_test:", y_test.shape)

print("\nFirst 10 feature names:")
print(feature_names[:10])

# 10) Plot class balance in TRAIN split (to confirm stratify worked)
train_counts = y_train.value_counts()
plt.figure(figsize=(6,4))
bars = plt.bar(train_counts.index.astype(str), train_counts.values)
plt.title("Train Split – Conservation Status (counts)")
plt.xlabel("Class"); plt.ylabel("Count")
for b in bars:
    plt.text(b.get_x()+b.get_width()/2, b.get_height(), int(b.get_height()),
             ha="center", va="bottom", fontsize=9)
plt.tight_layout()
plt.show()

```

```
# Keep these objects for the next step:
# X_train_proc, X_test_proc, y_train, y_test, preprocessor
```

### What to screenshot

- The printed **shapes** and **first 10 feature names**.
- The **bar chart** of train class counts.
- (If shown) the message about **dropping constant columns**.

✓ **Your turn: run this and take a screenshot. Then type CONTINUE to get the next step.**

-----  
User:

Continue

-----  
ChatGPT:

## Step 4 — Modeling

**Goal (kid-simple):** Train a few robots, see which one learns best. We'll try a **baseline**, **Logistic Regression**, and **Random Forest**. Pick the best by **F1 (macro)**, also show **Accuracy**.

### Colab-ready code

```
python
# === Step 4: Modeling ===
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import cross_validate, StratifiedKFold, train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix

# Settings
DATA_PATH = "crocodile_dataset.csv"
TARGET = "Conservation Status"
RANDOM_STATE = 42
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)
scoring = {"acc": "accuracy", "f1": "f1_macro"}

# --- Helper: if Step 3 variables aren't in memory, rebuild quietly ---
need_rebuild = any(name not in globals() for name in ["X_train", "X_test", "y_train", "y_test", "preprocessor"])
if need_rebuild:
    df = pd.read_csv(DATA_PATH)
    y = df[TARGET].astype(str)
    X = df.drop(columns=[TARGET])
    const_cols = [c for c in X.columns if X[c].nunique(dropna=True) <= 1]
    if const_cols: X = X.drop(columns=const_cols)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, stratify=y, random_state=RANDOM_STATE
    )
    num_cols = X_train.select_dtypes(include="number").columns.tolist()
    cat_cols = [c for c in X_train.columns if c not in num_cols]
    num_pipe = Pipeline([
        ("imputer", SimpleImputer(strategy="median")),
        ("scaler", StandardScaler())
    ])
    cat_pipe = Pipeline([
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("onehot", OneHotEncoder(handle_unknown="ignore", sparse=False))
    ])
    transformers = []
    if num_cols: transformers.append(("num", num_pipe, num_cols))
```

```

if cat_cols: transformers.append(("cat", cat_pipe, cat_cols))
preprocessor = ColumnTransformer(transformers=transformers, remainder="drop")

# --- Define candidate models (simple + strong) ---
candidates = {
    "Dummy (most_frequent)": DummyClassifier(strategy="most_frequent", random_state=RANDOM_STATE),
    "Logistic Regression": LogisticRegression(max_iter=2000, n_jobs=None),
    "Random Forest": RandomForestClassifier(n_estimators=300, random_state=RANDOM_STATE)
}

# --- Cross-validate each model using the full pipeline (preprocess inside) ---
rows = []
pipelines = {}
for name, clf in candidates.items():
    pipe = Pipeline([("prep", preprocessor), ("model", clf)])
    pipelines[name] = pipe
    cv_res = cross_validate(pipe, X_train, y_train, cv=cv, scoring=scoring, return_train_score=False)
    rows.append({
        "Model": name,
        "CV_F1_mean": np.mean(cv_res["test_f1"]),
        "CV_F1_std": np.std(cv_res["test_f1"]),
        "CV_Acc_mean": np.mean(cv_res["test_acc"]),
        "CV_Acc_std": np.std(cv_res["test_acc"]),
    })

cv_table = pd.DataFrame(rows).sort_values(by="CV_F1_mean", ascending=False).reset_index(drop=True)
print("=== Cross-Validation Results (5-fold) ===")
display(cv_table)

# --- Pick best by F1 (macro), fit on train, evaluate on test ---
best_name = cv_table.loc[0, "Model"]
best_model = pipelines[best_name].fit(X_train, y_train)
y_pred = best_model.predict(X_test)

test_acc = accuracy_score(y_test, y_pred)
test_f1 = f1_score(y_test, y_pred, average="macro")

print(f"\nBest model by CV: {best_name}")
print(f"TEST Accuracy: {test_acc:.4f}")
print(f"TEST F1 (macro): {test_f1:.4f}")

# --- Confusion Matrix plot (test) ---
labels = sorted(y_test.unique())
cm = confusion_matrix(y_test, y_pred, labels=labels)

plt.figure(figsize=(6,5))
im = plt.imshow(cm, aspect="auto")
plt.colorbar(im, fraction=0.046, pad=0.04)
plt.xticks(ticks=range(len(labels)), labels=labels, rotation=45, ha="right")
plt.yticks(ticks=range(len(labels)), labels=labels)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title(f"Confusion Matrix - {best_name} (Test)")
# annotate counts
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, str(cm[i, j]), ha="center", va="center", fontsize=9)
plt.tight_layout()
plt.show()

# Keep these objects for next steps:
# best_name, best_model, y_test, y_pred, test_acc, test_f1, cv_table

```

### What to screenshot

- The **Cross-Validation Results table** (sorted by F1).
- The printed **Best model + TEST Accuracy/F1** lines.
- The **Confusion Matrix** figure.

✅ **Your turn: run this and take a screenshot. Then type CONTINUE to get the next step.**

User:

Continue

ChatGPT:

## Step 5 — Evaluation

**Goal (kid-simple):** Check how good our best robot really is. Show **Accuracy** and **F1** on the test set, and how each class did.

### Colab-ready code

```
python
# === Step 5: Evaluation ===
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.metrics import classification_report, accuracy_score, f1_score, confusion_matrix
from sklearn.model_selection import StratifiedKFold, cross_validate, train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# --- Reuse from Step 4 if available; else rebuild quickly ---
DATA_PATH = "crocodile_dataset.csv"
TARGET = "Conservation Status"
RANDOM_STATE = 42
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)

def build_preprocessor(X_train):
    num_cols = X_train.select_dtypes(include="number").columns.tolist()
    cat_cols = [c for c in X_train.columns if c not in num_cols]
    num_pipe = Pipeline([("imputer", SimpleImputer(strategy="median")),
                        ("scaler", StandardScaler())])
    cat_pipe = Pipeline([("imputer", SimpleImputer(strategy="most_frequent")),
                        ("onehot", OneHotEncoder(handle_unknown="ignore", sparse=False))])

    tfs = []
    if num_cols: tfs.append(("num", num_pipe, num_cols))
    if cat_cols: tfs.append(("cat", cat_pipe, cat_cols))
    return ColumnTransformer(tfs, remainder="drop")

need_fit = any(name not in globals() for name in ["best_model", "best_name", "X_train", "X_test", "y_train", "y_test"])
if need_fit:
    # Minimal rebuild + choose best by CV F1
    df_eval = pd.read_csv(DATA_PATH)
    y = df_eval[TARGET].astype(str)
    X = df_eval.drop(columns=[TARGET])
    const_cols = [c for c in X.columns if X[c].nunique(dropna=True) <= 1]
    if const_cols: X = X.drop(columns=const_cols)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, stratify=y, random_state=RANDOM_STATE
    )
    preprocessor = build_preprocessor(X_train)

    candidates = {
        "Dummy (most_frequent)": DummyClassifier(strategy="most_frequent", random_state=RANDOM_STATE),
        "Logistic Regression": LogisticRegression(max_iter=2000),
        "Random Forest": RandomForestClassifier(n_estimators=300, random_state=RANDOM_STATE),
    }

    rows, pipelines = [], {}
    for name, clf in candidates.items():
        pipe = Pipeline([("prep", preprocessor), ("model", clf)])
        pipelines[name] = pipe
        res = cross_validate(pipe, X_train, y_train, cv=cv,
                            scoring={"f1": "f1_macro", "acc": "accuracy"})
        rows.append({"Model": name,
                    "CV_F1_mean": np.mean(res["test_f1"]),
                    "CV_F1_std": np.std(res["test_f1"]),
                    "CV_Acc_mean": np.mean(res["test_acc"]),
                    "CV_Acc_std": np.std(res["test_acc"])})
```

```

cv_table = pd.DataFrame(rows).sort_values("CV_F1_mean", ascending=False).reset_index(drop=True)
best_name = cv_table.loc[0, "Model"]
best_model = pipelines[best_name].fit(X_train, y_train)

# Predictions on test
y_pred = best_model.predict(X_test)

test_acc = accuracy_score(y_test, y_pred)
test_f1 = f1_score(y_test, y_pred, average="macro")

print("=== Test Metrics ===")
print(f"Best model: {best_name}")
print(f"Accuracy: {test_acc:.4f}")
print(f"F1 (macro): {test_f1:.4f}")

# Classification report (as a nice table)
report = classification_report(y_test, y_pred, output_dict=True, zero_division=0)
rep_df = pd.DataFrame(report).T
rep_df = rep_df.rename(columns={"precision": "Precision", "recall": "Recall", "f1-score": "F1", "support": "Support"})
# Keep per-class rows + macro/weighted avg
keep_rows = [r for r in rep_df.index if r not in ["accuracy"]]
rep_df_display = rep_df.loc[keep_rows].copy()
display(rep_df_display.round(3))

# Per-class F1 bar chart
per_class = rep_df_display.drop(index=["macro avg", "weighted avg"], errors="ignore")
if not per_class.empty:
    plt.figure(figsize=(6,4))
    bars = plt.bar(per_class.index.astype(str), per_class["F1"].values)
    plt.title(f"Per-Class F1 – {best_name} (Test)")
    plt.xlabel("Class"); plt.ylabel("F1")
    for b, v in zip(bars, per_class["F1"].values):
        plt.text(b.get_x()+b.get_width()/2, v, f"{v:.2f}", ha="center", va="bottom", fontsize=9)
    plt.ylim(0, 1.05)
    plt.tight_layout()
    plt.show()

# Normalized confusion matrix (percent by true class)
labels = sorted(y_test.unique())
cm = confusion_matrix(y_test, y_pred, labels=labels)
cm_norm = cm / cm.sum(axis=1, keepdims=True)
plt.figure(figsize=(6,5))
im = plt.imshow(cm_norm, aspect="auto")
plt.colorbar(im, fraction=0.046, pad=0.04)
plt.xticks(ticks=range(len(labels)), labels=labels, rotation=45, ha="right")
plt.yticks(ticks=range(len(labels)), labels=labels)
plt.xlabel("Predicted"); plt.ylabel("True")
plt.title(f"Confusion Matrix (row-normalized) – {best_name}")
# annotate with percentages
for i in range(cm_norm.shape[0]):
    for j in range(cm_norm.shape[1]):
        plt.text(j, i, f"{cm_norm[i, j]*100:.1f}%", ha="center", va="center", fontsize=9)
plt.tight_layout()
plt.show()

# (Optional) Compare CV vs Test for the chosen model
try:
    row = cv_table[cv_table["Model"] == best_name].iloc[0]
    compare = pd.DataFrame({
        "Model": [best_name],
        "CV_F1_mean": [row["CV_F1_mean"]],
        "CV_F1_std": [row["CV_F1_std"]],
        "CV_Acc_mean": [row["CV_Acc_mean"]],
        "CV_Acc_std": [row["CV_Acc_std"]],
        "TEST_F1_macro": [test_f1],
        "TEST_Accuracy": [test_acc],
    })
    print("\n=== CV vs TEST (chosen model) ===")
    display(compare.round(3))
except Exception as e:
    # If cv_table not available (e.g., user skipped Step 4), silently skip
    pass

# Keep: best_name, best_model, y_test, y_pred, test_acc, test_f1, rep_df_display

```

## What to screenshot



- The **Test Metrics** (Accuracy and F1).
- The **classification report table**.
- The **Per-Class F1 bar chart**.
- The **row-normalized confusion matrix**.
- (Nice-to-have) The small **CV vs TEST** comparison table.

✓ **Your turn: run this and take a screenshot. Then type CONTINUE to get the next step.**

User:

Continue

ChatGPT:

## Step 6 — Insights & Next Steps

**Goal (kid-simple):** See which features mattered, where mistakes happen, and what to fix next.

**Colab-ready code**

```
python
# === Step 6: Insights & Next Steps ===
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import StratifiedKFold, learning_curve, train_test_split, cross_validate
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# --- Reuse trained best_model; rebuild quickly if missing ---
DATA_PATH = "crocodile_dataset.csv"
TARGET = "Conservation Status"
RANDOM_STATE = 42

def build_preprocessor(X_train):
    num_cols = X_train.select_dtypes(include="number").columns.tolist()
    cat_cols = [c for c in X_train.columns if c not in num_cols]
    num_pipe = Pipeline([("imputer", SimpleImputer(strategy="median")),
                          ("scaler", StandardScaler())])
    cat_pipe = Pipeline([("imputer", SimpleImputer(strategy="most_frequent")),
                          ("onehot", OneHotEncoder(handle_unknown="ignore", sparse=False))])

    tfs = []
    if num_cols: tfs.append(("num", num_pipe, num_cols))
    if cat_cols: tfs.append(("cat", cat_pipe, cat_cols))
    return ColumnTransformer(tfs, remainder="drop")

def quick_fit_best():
    df = pd.read_csv(DATA_PATH)
    y = df[TARGET].astype(str)
    X = df.drop(columns=[TARGET])
    const_cols = [c for c in X.columns if X[c].nunique(dropna=True) <= 1]
    if const_cols: X = X.drop(columns=const_cols)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, stratify=y, random_state=RANDOM_STATE
    )
    preprocessor = build_preprocessor(X_train)
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)
    candidates = {
        "Dummy (most_frequent)": DummyClassifier(random_state=RANDOM_STATE),
        "Logistic Regression": LogisticRegression(max_iter=2000),
        "Random Forest": RandomForestClassifier(n_estimators=300, random_state=RANDOM_STATE),
    }
    rows, pipelines = [], {}
```

```

for name, clf in candidates.items():
    pipe = Pipeline([("prep", preprocessor), ("model", clf)])
    pipelines[name] = pipe
    res = cross_validate(pipe, X_train, y_train, cv=cv,
                        scoring={"f1": "f1_macro", "acc": "accuracy"})
    rows.append({"Model": name,
                "CV_F1_mean": sum(res["test_f1"])/len(res["test_f1"]),
                "CV_Acc_mean": sum(res["test_acc"])/len(res["test_acc"])})
cv_table = pd.DataFrame(rows).sort_values("CV_F1_mean", ascending=False).reset_index(drop=True)
best_name = cv_table.loc[0, "Model"]
best_model = pipelines[best_name].fit(X_train, y_train)
return best_model, best_name, X_train, X_test, y_train, y_test

if any(n not in globals() for n in ["best_model", "best_name", "X_train", "X_test", "y_train", "y_test"]):
    best_model, best_name, X_train, X_test, y_train, y_test = quick_fit_best()

# --- Helper: get feature names after ColumnTransformer (handles OHE) ---
def get_feature_names_from_ct(ct: ColumnTransformer):
    output_feats = []
    for name, trans, cols in ct.transformers_:
        if name == "remainder" and trans == "drop":
            continue
        # If it's a Pipeline, look for OneHotEncoder inside; else just use original names.
        if hasattr(trans, "named_steps") and "onehot" in trans.named_steps and hasattr(trans.named_steps["onehot"], "get_feature_names_out"):
            ohe = trans.named_steps["onehot"]
            output_feats.extend(list(ohe.get_feature_names_out(cols)))
        else:
            # numeric pipeline or other: keep original names
            output_feats.extend(list(cols))
    return output_feats

# --- 1) Feature importance (works for RF or LR) ---
preproc = best_model.named_steps["prep"]
clf = best_model.named_steps["model"]
feat_names = get_feature_names_from_ct(preproc)

# Importance vector:
importances = None
if hasattr(clf, "feature_importances_"): # RandomForest
    importances = pd.Series(clf.feature_importances_, index=feat_names)
elif hasattr(clf, "coef_"): # LogisticRegression (multiclass OK)
    # average absolute coefficients across classes
    coefs = abs(pd.DataFrame(clf.coef_, columns=feat_names))
    importances = coefs.mean(axis=0)
else:
    print("This model does not expose feature importances/coefficients.")
    importances = pd.Series([0]*len(feat_names), index=feat_names)

imp_df = (importances.sort_values(ascending=False)
          .rename("importance")
          .reset_index()
          .rename(columns={"index": "feature"}))
topk = 15 if len(imp_df) >= 15 else len(imp_df)

plt.figure(figsize=(7, 6))
plot_df = imp_df.head(topk).iloc[::-1] # reverse for barh ascending
plt.barh(plot_df["feature"], plot_df["importance"])
plt.title(f"Top {topk} Features - {best_name}")
plt.xlabel("Importance (relative)")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()

# --- 2) Where are errors? (per-class report + most confused pairs) ---
y_pred = best_model.predict(X_test)
report = pd.DataFrame(classification_report(y_test, y_pred, output_dict=True, zero_division=0)).T
per_class = report.drop(index=["accuracy", "macro avg", "weighted avg"], errors="ignore").copy()
per_class = per_class.rename(columns={"precision": "Precision", "recall": "Recall", "f1-score": "F1", "support": "Support"})
per_class = per_class.sort_values("F1") # worst at top
print("=== Per-class metrics (Test) - Worst first ===")
display(per_class.round(3))

# Most confused pairs (top 5)
labels = sorted(y_test.unique())
cm = confusion_matrix(y_test, y_pred, labels=labels)
cm_df = pd.DataFrame(cm, index=pd.Index(labels, name="True"), columns=pd.Index(labels, name="Pred"))
pairs = (cm_df.stack().rename("count").reset_index())

```

```

pairs = pairs[pairs["True"] != pairs["Pred"]].sort_values("count", ascending=False)
print("\n=== Most confused class pairs (Test) ===")
display(pairs.head(5))

# --- 3) Do we need more data? (simple learning curve on F1) ---
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)
sizes = [0.2, 0.4, 0.6, 0.8, 1.0]
train_sizes, train_scores, val_scores = learning_curve(
    estimator=best_model,
    X=X_train, y=y_train,
    train_sizes=sizes, cv=cv, scoring="f1_macro", exploit_incremental_learning=False
)

train_mean = train_scores.mean(axis=1)
val_mean = val_scores.mean(axis=1)

plt.figure(figsize=(6,4))
plt.plot(train_sizes, train_mean, marker="o", label="Train F1")
plt.plot(train_sizes, val_mean, marker="o", label="CV F1")
plt.title(f"Learning Curve - {best_name}")
plt.xlabel("Training set size")
plt.ylabel("F1 (macro)")
plt.legend()
plt.tight_layout()
plt.show()

# --- 4) Auto-suggestions (tiny, data-driven) ---
imbalance_ratio = (y_train.value_counts().max() / y_train.value_counts().min()) if y_train.nunique() > 1 else 1.0
print("=== Next Steps (Suggestions) ===")
if imbalance_ratio > 1.5:
    print("- Classes look imbalanced. Try class_weight='balanced' (LogReg) or tune class_weight in RandomForest.")
worst = per_class.head(1)
if not worst.empty and worst["F1"].iloc[0] < 0.7:
    print(f"- The '{worst.index[0]}' class is weak (F1 ~ {worst['F1'].iloc[0]:.2f}). Add examples or class-specific features")
print("- Try simple hyperparameter tuning (n_estimators/max_depth for RF; C/penalty for LogReg).")
print("- Check top confused pairs above and add features that separate those two classes.")

```

### What to screenshot

- The **Top Features** horizontal bar chart.
- The **Per-class metrics table** (worst first).
- The **Most confused class pairs** table.
- The **Learning Curve** plot.

✅ **Your turn: run this and take a screenshot. Then type CONTINUE to get the next step.**

-----  
User:

Continue

-----  
ChatGPT:

## Step 7 — Wrap-Up (Save Deliverables)

**Goal (kid-simple):** Save neat files (tables + plots) so your report is easy to build.

### Colab-ready code

```

python
# === Step 7: Wrap-Up / Save Deliverables ===
import os
import json
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.metrics import (
    classification_report, accuracy_score, f1_score, confusion_matrix
)

```

```

from sklearn.model_selection import (
    StratifiedKFold, cross_validate, train_test_split, learning_curve
)
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# --- Settings ---
DATA_PATH = "crocodile_dataset.csv"
TARGET = "Conservation Status"
RANDOM_STATE = 42
OUTDIR = "outputs"
os.makedirs(OUTDIR, exist_ok=True)

# --- Helpers to rebuild if notebook was restarted ---
def build_preprocessor(X_train):
    num_cols = X_train.select_dtypes(include="number").columns.tolist()
    cat_cols = [c for c in X_train.columns if c not in num_cols]
    num_pipe = Pipeline([("imputer", SimpleImputer(strategy="median")),
                          ("scaler", StandardScaler())])
    cat_pipe = Pipeline([("imputer", SimpleImputer(strategy="most_frequent")),
                          ("onehot", OneHotEncoder(handle_unknown="ignore", sparse=False))])

    tfs = []
    if num_cols: tfs.append(("num", num_pipe, num_cols))
    if cat_cols: tfs.append(("cat", cat_pipe, cat_cols))
    return ColumnTransformer(tfs, remainder="drop")

def quick_fit_best_and_cv():
    df = pd.read_csv(DATA_PATH)
    y = df[TARGET].astype(str)
    X = df.drop(columns=[TARGET])
    const_cols = [c for c in X.columns if X[c].nunique(dropna=True) <= 1]
    if const_cols:
        X = X.drop(columns=const_cols)

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, stratify=y, random_state=RANDOM_STATE
    )
    preprocessor = build_preprocessor(X_train)
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)

    candidates = {
        "Dummy (most frequent)": DummyClassifier(strategy="most_frequent", random_state=RANDOM_STATE),
        "Logistic Regression": LogisticRegression(max_iter=2000),
        "Random Forest": RandomForestClassifier(n_estimators=300, random_state=RANDOM_STATE),
    }

    rows, pipelines = [], {}
    for name, clf in candidates.items():
        pipe = Pipeline([("prep", preprocessor), ("model", clf)])
        pipelines[name] = pipe
        res = cross_validate(pipe, X_train, y_train, cv=cv,
                             scoring={"f1": "f1_macro", "acc": "accuracy"})
        rows.append({
            "Model": name,
            "CV_F1_mean": np.mean(res["test_f1"]),
            "CV_F1_std": np.std(res["test_f1"]),
            "CV_Acc_mean": np.mean(res["test_acc"]),
            "CV_Acc_std": np.std(res["test_acc"]),
        })

    cv_table = pd.DataFrame(rows).sort_values("CV_F1_mean", ascending=False).reset_index(drop=True)
    best_name = cv_table.loc[0, "Model"]
    best_model = pipelines[best_name].fit(X_train, y_train)
    return best_model, best_name, X_train, X_test, y_train, y_test, cv_table

# Reuse from earlier steps if present; else rebuild
need = any(v not in globals() for v in ["best_model", "best_name", "X_train", "X_test", "y_train", "y_test"])
if need:
    best_model, best_name, X_train, X_test, y_train, y_test, cv_table = quick_fit_best_and_cv()

# If CV table missing (e.g., came from earlier run), try to recreate quickly
if "cv_table" not in globals():

```

```

    _bm, _bn, _a, _b, _c, _d, cv_table = quick_fit_best_and_cv()

# --- 1) Test predictions and metrics ---
y_pred = best_model.predict(X_test)
test_acc = accuracy_score(y_test, y_pred)
test_f1 = f1_score(y_test, y_pred, average="macro")

# Save metrics
pd.DataFrame({"metric":["accuracy","f1_macro"], "value":[test_acc, test_f1]}).to_csv(
    f"{OUTDIR}/test_metrics.csv", index=False
)

# Save classification report
report = classification_report(y_test, y_pred, output_dict=True, zero_division=0)
pd.DataFrame(report).T.to_csv(f"{OUTDIR}/classification_report.csv")

# Save predictions
pd.DataFrame({"y_true": y_test.reset_index(drop=True),
              "y_pred": pd.Series(y_pred)}).to_csv(f"{OUTDIR}/predictions.csv", index=False)

# Save CV table
cv_table.to_csv(f"{OUTDIR}/cv_results.csv", index=False)

# --- 2) Plots to PNG ---
# Confusion matrix (counts)
labels = sorted(y_test.unique())
cm = confusion_matrix(y_test, y_pred, labels=labels)

plt.figure(figsize=(6,5))
im = plt.imshow(cm, aspect="auto")
plt.colorbar(im, fraction=0.046, pad=0.04)
plt.xticks(range(len(labels)), labels, rotation=45, ha="right")
plt.yticks(range(len(labels)), labels)
plt.xlabel("Predicted"); plt.ylabel("True")
plt.title(f"Confusion Matrix - {best_name} (Test)")
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, str(cm[i, j]), ha="center", va="center", fontsize=9)
plt.tight_layout()
plt.savefig(f"{OUTDIR}/confusion_matrix_counts.png", dpi=160)
plt.close()

# Normalized CM
cm_norm = cm / cm.sum(axis=1, keepdims=True)
plt.figure(figsize=(6,5))
im = plt.imshow(cm_norm, aspect="auto")
plt.colorbar(im, fraction=0.046, pad=0.04)
plt.xticks(range(len(labels)), labels, rotation=45, ha="right")
plt.yticks(range(len(labels)), labels)
plt.xlabel("Predicted"); plt.ylabel("True")
plt.title(f"Confusion Matrix (row-normalized) - {best_name}")
for i in range(cm_norm.shape[0]):
    for j in range(cm_norm.shape[1]):
        plt.text(j, i, f"{cm_norm[i, j]*100:.1f}%", ha="center", va="center", fontsize=9)
plt.tight_layout()
plt.savefig(f"{OUTDIR}/confusion_matrix_normalized.png", dpi=160)
plt.close()

# Per-class F1 bar chart
rep_df = pd.DataFrame(report).T.rename(columns={"precision":"Precision","recall":"Recall","f1-score":"F1"})
per_class = rep_df.drop(index=["accuracy","macro avg","weighted avg"], errors="ignore")
if not per_class.empty:
    plt.figure(figsize=(6,4))
    vals = per_class["F1"].values
    idxs = per_class.index.astype(str)
    bars = plt.bar(idxs, vals)
    plt.title(f"Per-Class F1 - {best_name} (Test)")
    plt.xlabel("Class"); plt.ylabel("F1")
    for b, v in zip(bars, vals):
        plt.text(b.get_x()+b.get_width()/2, v, f"{v:.2f}", ha="center", va="bottom", fontsize=9)
    plt.ylim(0, 1.05)
    plt.tight_layout()
    plt.savefig(f"{OUTDIR}/per_class_f1.png", dpi=160)
    plt.close()

# Learning curve (F1 macro)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)

```

```

sizes = [0.2, 0.4, 0.6, 0.8, 1.0]
train_sizes, train_scores, val_scores = learning_curve(
    estimator=best_model, X=X_train, y=y_train,
    train_sizes=sizes, cv=cv, scoring="f1_macro", exploit_incremental_learning=False
)
plt.figure(figsize=(6,4))
plt.plot(train_sizes, train_scores.mean(axis=1), marker="o", label="Train F1")
plt.plot(train_sizes, val_scores.mean(axis=1), marker="o", label="CV F1")
plt.title(f"Learning Curve - {best_name}")
plt.xlabel("Training set size"); plt.ylabel("F1 (macro)")
plt.legend()
plt.tight_layout()
plt.savefig(f"{OUTDIR}/learning_curve.png", dpi=160)
plt.close()

# Top features (works for RF/LR)
def get_feature_names_from_ct(ct: ColumnTransformer):
    out = []
    for name, trans, cols in ct.transformers_:
        if name == "remainder" and trans == "drop":
            continue
        if hasattr(trans, "named_steps") and "onehot" in trans.named_steps and hasattr(trans.named_steps["onehot"], "get_feature_names_out"):
            out.extend(list(trans.named_steps["onehot"].get_feature_names_out(cols)))
        else:
            out.extend(list(cols))
    return out

preproc = best_model.named_steps["prep"]
clf = best_model.named_steps["model"]
feat_names = get_feature_names_from_ct(preproc)

if hasattr(clf, "feature_importances_"):
    importances = pd.Series(clf.feature_importances_, index=feat_names)
elif hasattr(clf, "coef_"):
    coefs = np.abs(clf.coef_)
    if coefs.ndim == 2: # multiclass
        importances = pd.Series(coefs.mean(axis=0), index=feat_names)
    else:
        importances = pd.Series(np.abs(coefs), index=feat_names)
else:
    importances = pd.Series([0]*len(feat_names), index=feat_names)

imp_df = importances.sort_values(ascending=False).rename("importance").reset_index().rename(columns={"index": "feature"})
imp_df.to_csv(f"{OUTDIR}/top_features.csv", index=False)

topk = min(15, len(imp_df))
plt.figure(figsize=(7,6))
plot_df = imp_df.head(topk).iloc[::-1]
plt.barh(plot_df["feature"], plot_df["importance"])
plt.title(f"Top {topk} Features - {best_name}")
plt.xlabel("Importance (relative)"); plt.ylabel("Feature")
plt.tight_layout()
plt.savefig(f"{OUTDIR}/top_features.png", dpi=160)
plt.close()

# --- 3) Tiny README for your report ---
readme = {
    "goal": "Predict crocodile Conservation Status.",
    "target": TARGET,
    "metrics": {"test_accuracy": round(float(test_acc), 4),
               "test_f1_macro": round(float(test_f1), 4)},
    "best_model": str(best_name),
    "files": sorted(os.listdir(OUTDIR))
}
with open(f"{OUTDIR}/README.json", "w") as f:
    json.dump(readme, f, indent=2)

print("Saved files in 'outputs/':")
for f in sorted(os.listdir(OUTDIR)):
    print(" -", f)

```

## What to screenshot

- A file list of the **outputs/** folder showing PNG/CSV/JSON files.
- (Pick any one figure) e.g., **confusion\_matrix\_normalized.png** opened in Colab.

✅ Your turn: run this and take a screenshot. Then type CONTINUE to get the next step.

---













































































































