```
import os
import json
import time
from collections import Counter
from typing import List, Tuple

import joblib
import numpy as np
import pandas as pd
import streamlit as st
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    confusion_matrix,
    roc_curve,
    auc,
    precision_recall_curve,
    PrecisionRecallDisplay,
)

st.set_page_config(page_title="Spam Classifier — Visualizations", layout="wide")
```

```
def ts() -> str:
    return time.strftime("%Y%m%d-%H%M%S")


@st.cache_data(show_spinner=False)
def load_csv(path: str) -> pd.DataFrame:
    return pd.read_csv(path)


@st.cache_data(show_spinner=False)
def list_datasets() -> List[str]:
    paths: List[str] = []
    for root in ("datasets", os.path.join("datasets", "processed")):
        if os.path.isdir(root):
            for name in os.listdir(root):
                p = os.path.join(root, name)
                if name.lower().endswith(".csv") and os.path.isfile(p):
                    paths.append(p)
    return sorted(paths)


def infer_cols(df: pd.DataFrame) -> Tuple[str, str]:
    # Try to infer label/text columns
```

```
    cols = list(df.columns)
    label_candidates = [c for c in cols if c.lower() in ("label", "target", "col_0")]
    text_candidates = [c for c in cols if c.lower() in ("text", "message", "text_clean",
"col_1")]
    label = label_candidates[0] if label_candidates else cols[0]
    text = text_candidates[0] if text_candidates else cols[-1]
    return label, text



def token_topn(series: pd.Series, topn: int) -> List[Tuple[str, int]]:
    counter: Counter = Counter()
    for s in series.astype(str):
        counter.update(s.split())
    return counter.most_common(topn)



@st.cache_resource(show_spinner=False)
def load_artifacts(models_dir: str):
    vec = joblib.load(os.path.join(models_dir, "spam_tfidf_vectorizer.joblib"))
    clf = joblib.load(os.path.join(models_dir, "spam_logreg_model.joblib"))
    pos, neg = "spam", "ham"
    meta_p = os.path.join(models_dir, "spam_label_mapping.json")
    if os.path.exists(meta_p):
        try:
```

```
                with open(meta_p, "r", encoding="utf-8") as f:
                    meta = json.load(f)
                    pos = meta.get("positive", pos)
                    neg = meta.get("negative", neg)
            except Exception:
                pass
        return vec, clf, pos, neg



def label_to_int(series: pd.Series, pos_label: str = "spam") -> np.ndarray:
        s = series.astype(str).str.lower()
        return (s == pos_label.lower()).astype(int).values



# Lightweight normalization to match training text_clean behavior for live inference
import re
URL_RE = re.compile(r"https?://\S+|www\.\S+", re.IGNORECASE)
EMAIL_RE = re.compile(r"\b[\w\.-]+@[\w\.-]+\.[a-zA-Z]{2,}\b")
PHONE_RE = re.compile(r"\b(?:\+?\d[\d\-\s]{7,}\d)\b")

def normalize_text(text: str, keep_numbers: bool = False) -> str:
        if not isinstance(text, str):
            text = "" if text is None else str(text)
        t = text.lower()
```

```
    t = URL_RE.sub("<URL>", t)
    t = EMAIL_RE.sub("<EMAIL>", t)
    t = PHONE_RE.sub("<PHONE>", t)
    if not keep_numbers:
        t = re.sub(r"\d+", "<NUM>", t)
    t = re.sub(r"[^\w\s<>]", " ", t)
    t = re.sub(r"\s+", " ", t).strip()
    return t


def main():
    st.title("Spam/Ham Classifier — Phase 4 Visualizations")
    st.caption("Interactive dashboard for data distribution, token patterns, and model performance")


    # Sidebar: data and artifacts
    with st.sidebar:
        st.header("Inputs")
        datasets = list_datasets()
        ds_path = st.selectbox("Dataset CSV", datasets, index=datasets.index("datasets/processed/sms_spam_clean.csv") if "datasets/processed/sms_spam_clean.csv" in datasets else 0)
        df = load_csv(ds_path)
        label_col, text_col = infer_cols(df)
        label_col = st.selectbox("Label column", options=list(df.columns), index=list(df.columns).index(label_col))
```

```
        text_col = st.selectbox("Text column", options=list(df.columns),
index=list(df.columns).index(text_col))


        models_dir = st.text_input("Models dir", value="models")
        test_size = st.slider("Test size", min_value=0.1, max_value=0.4, value=0.2,
step=0.05)
        seed = st.number_input("Seed", min_value=0, value=42, step=1)
        threshold = st.slider("Decision threshold", min_value=0.1, max_value=0.9,
value=0.5, step=0.01)


    st.subheader("Data Overview")
    c1, c2 = st.columns(2)
    with c1:
        st.write("Class distribution")
        counts = df[label_col].value_counts().sort_index()
        st.bar_chart(counts)
    with c2:
        st.write("Token replacements in cleaned text (approximate)")
        sample = df[text_col].astype(str)
        repl = {
            "<URL>": sample.str.count(r"<URL>").sum(),
            "<EMAIL>": sample.str.count(r"<EMAIL>").sum(),
            "<PHONE>": sample.str.count(r"<PHONE>").sum(),
            "<NUM>": sample.str.count(r"<NUM>").sum(),
        }
```

```
            st.table(pd.DataFrame.from_dict(repl, orient="index", columns=["count"]))


        st.subheader("Top Tokens by Class")
        topn = st.slider("Top-N tokens", min_value=10, max_value=40, value=20,
step=5)
        col_a, col_b = st.columns(2)
        for label, col in [(counts.index[0], col_a), (counts.index[-1], col_b)]:
            with col:
                st.write(f"Class: {label}")
                top = token_topn(df.loc[df[label_col] == label, text_col], topn)
                if top:
                    toks, freqs = zip(*top)
                    fig, ax = plt.subplots(figsize=(6, 4))
                    sns.barplot(x=list(freqs), y=list(toks), ax=ax, palette="viridis")
                    ax.set_xlabel("frequency"); ax.set_ylabel("token")
                    st.pyplot(fig)
                else:
                    st.info("No tokens found.")


        # Model-based visuals
        st.subheader("Model Performance (Test)")
        if os.path.exists(os.path.join(models_dir, "spam_tfidf_vectorizer.joblib")) and
os.path.exists(os.path.join(models_dir, "spam_logreg_model.joblib")):
            vec, clf, pos_label, neg_label = load_artifacts(models_dir)
            X = df[text_col].astype(str).fillna("")
```

```
            y = label_to_int(df[label_col], pos_label=pos_label)
            Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=test_size,
random_state=seed, stratify=y)
            Xte_vec = vec.transform(Xte)
            proba = clf.predict_proba(Xte_vec)[:, 1]
            pred = (proba >= threshold).astype(int)

            # Confusion matrix
            cm = confusion_matrix(yte, pred)
            cm_df = pd.DataFrame(cm, index=["true_0","true_1"],
columns=["pred_0","pred_1"])
            st.write("Confusion matrix")
            st.dataframe(cm_df)

            # ROC/PR curves
            fpr, tpr, _ = roc_curve(yte, proba)
            roc_auc = auc(fpr, tpr)
            prec, rec, _ = precision_recall_curve(yte, proba)
            pr_fig, pr_ax = plt.subplots(1, 2, figsize=(10, 4))
            pr_ax[0].plot(fpr, tpr, label=f"AUC={roc_auc:.3f}")
            pr_ax[0].plot([0,1],[0,1], linestyle="--", color="gray")
            pr_ax[0].set_title("ROC")
            pr_ax[0].set_xlabel("FPR"); pr_ax[0].set_ylabel("TPR")
            PrecisionRecallDisplay(precision=prec, recall=rec).plot(ax=pr_ax[1])
            pr_ax[1].set_title("Precision-Recall")
```

```
        st.pyplot(pr_fig)


        # Threshold sweep small table
        st.write("Threshold sweep (precision/recall/f1)")
        ths = np.round(np.linspace(0.3, 0.8, 11), 3)
        rows = []
        for t in ths:
            p = (proba >= t).astype(int)
            from sklearn.metrics import precision_score, recall_score, f1_score


            rows.append({
                "threshold": t,
                "precision": float(precision_score(yte, p, zero_division=0)),
                "recall": float(recall_score(yte, p, zero_division=0)),
                "f1": float(f1_score(yte, p, zero_division=0)),
            })
        st.dataframe(pd.DataFrame(rows))


        # Live Inference
        st.subheader("Live Inference")
        # Provide two quick examples to try
        ex_spam = "Free entry in 2 a wkly comp to win cash now! Call +44 906-170-
1461 to claim prize"
        ex_ham = "Ok, I'll see you at 7 pm for dinner. Thanks!"
```

```
    c_ex1, c_ex2 = st.columns(2)
    with c_ex1:
        if st.button("Use spam example"):
            st.session_state["input_text"] = ex_spam
    with c_ex2:
        if st.button("Use ham example"):
            st.session_state["input_text"] = ex_ham

    # Text area bound to session_state so examples populate it
    if "input_text" not in st.session_state:
        st.session_state["input_text"] = ""
    user_text = st.text_area("Enter a message to classify", key="input_text")

    if st.button("Predict"):
        if user_text.strip():
            cleaned = normalize_text(user_text)
            with st.expander("Show normalized text", expanded=False):
                st.code(cleaned)
            X_single = vec.transform([cleaned])
            prob = float(clf.predict_proba(X_single)[:, 1][0])
            pred_label = pos_label if prob >= threshold else neg_label
            st.success(f"Prediction: {pred_label}    |    spam-prob = {prob:.4f}
(threshold = {threshold:.2f})")
```

```
                    # Probability bar (0..1) with threshold marker
                    fig_g, ax_g = plt.subplots(figsize=(6, 0.6))
                    ax_g.barh([0], [prob], color="#d62728" if pred_label == pos_label
else "#1f77b4")
                    ax_g.axvline(threshold, color="black", linestyle="--", linewidth=1)
                    ax_g.set_xlim(0, 1)
                    ax_g.set_yticks([])
                    ax_g.set_xlabel("spam probability")
                    ax_g.text(min(prob + 0.02, 0.98), 0, f"{prob:.2f}", va="center")
                    st.pyplot(fig_g)
                else:
                    st.info("Please enter a non-empty message.")


    else:
        st.info("Model artifacts not found in 'models/'. Train the model first to
enable performance plots.")


if __name__ == "__main__":
    main()
```