Automated Clinical Text Classification with Robust Machine Learning for Pharmaceutical Data Curation

Weineng Zhou, Kunwan Chen
Clinflash, Nanjing, China

ABSTRACT

In clinical trial data management, table-of-contents (TOC) Excel files must be annotated with CDISC dataset identifiers (e.g., ADSL, ADAE, ADLB) before downstream statistical analysis. At Clinflash, labeling thousands of Chinese-language table titles across dozens of studies was performed manually—a process requiring hours per study and producing inconsistent results between reviewers. This paper presents a lightweight Python tool that automates TOC dataset labeling using TF-IDF text vectorization (ngram_range=(1,2), max_features=5000) and multinomial logistic regression (lbfgs solver, max_iter=1000). The tool consists of two scripts: predict_dataset_cn.py, a one-click production tool that trains on all available labeled TOC Excel files in the working directory and predicts dataset labels for new studies via a tkinter file-selection dialog, outputting a _dataset_filled.xlsx file; and model.py, an evaluation script that performs stratified 80/20 train-test splitting, binary classification assessment (largest class vs. rest), and exports nine clinical metrics to evaluation_report.xlsx alongside four publication-quality plots (ROC, PR, calibration curves, confusion matrix). Trained on 1,849 rows aggregated from 14 clinical study TOC files spanning 60 CDISC dataset labels, the production model handles multi-class prediction directly. On a held-out binary evaluation, the model achieves accuracy=0.8859, ROC-AUC=0.9349, sensitivity=0.6522, and specificity=0.9398. A standalone Windows executable (predict_dataset_cn.exe), built with PyInstaller, allows non-technical users to run the tool without installing Python or any dependencies.

KEYWORDS: TF-IDF, Logistic Regression, Text Classification, CDISC, Table of Contents, Clinical Data Curation, Python, tkinter GUI, PyInstaller

1. INTRODUCTION

In pharmaceutical statistical programming, every clinical study produces a table-of-contents (TOC) Excel file listing all planned tables, listings, and figures (TLFs). Each entry includes a descriptive TITLE (e.g., “TEAE 按照 MedDRA 系统器官分类和首选术语的发生率”) and must be assigned a corresponding CDISC DATASET label (e.g., ADAE, ADSL, ADLB) that identifies which analysis dataset the table draws from. At Clinflash, this labeling task was performed manually by statistical programmers reviewing each TITLE and selecting the appropriate DATASET from memory or a reference document. For a typical study with 100–200 tables, this took 30–60 minutes of focused effort; across 14 studies in our training corpus (1,849 rows total, spanning 60 unique DATASET values), inconsistencies between reviewers were common, particularly for edge cases such as tables drawing from multiple datasets (e.g., “ADSL, ADTTE”). The task is repetitive, rule-based, and well-suited to automation via supervised text classification.

Before developing this tool, we relied on the following approaches, each with practical drawbacks:

- Manual lookup: Programmers searched a reference spreadsheet or relied on memory to match each TITLE to its DATASET. For 100+ tables per study, this was tedious and prone to fatigue-related errors; different programmers often assigned different labels to the same TITLE pattern.
- Keyword-based SAS macros: We attempted keyword-matching logic (e.g., IF INDEX(TITLE, “不良事件”) THEN DATASET=“ADAE”), but these required continuous maintenance as new studies introduced unseen TITLE phrasing, and the rules grew unwieldy across 60 target labels.
- Cross-project inconsistency: A TITLE containing “实验室检查” (laboratory tests) might be labeled ADLB by one programmer and ADHY (hy’s law) by another depending on study context, creating downstream dataset merging problems.

- Scaling bottleneck: As Clinflash's portfolio grew to dozens of concurrent studies, the cumulative labeling effort became a measurable drag on statistical programming turnaround time.

Supervised text classification can automate this task: learn the TITLE-to-DATASET mapping from previously labeled studies, then apply it to new ones. However, off-the-shelf ML scripts require adaptation for routine pharmaceutical use:

- Clinical evaluation metrics: In addition to accuracy and F1, pharmaceutical applications require sensitivity, specificity, PPV, and NPV to assess the risk of mislabeling safety-critical datasets (e.g., confusing ADAE with ADLB). Generic ML scripts typically report only accuracy and AUC.
- Class imbalance: In our training data, the top 5 DATASET labels (ADLB, ADAE, ADSL, ADMI, ADQS) account for 62% of rows, while 14 of 60 labels appear only once. Training a usable multi-class classifier on this distribution requires handling rare labels gracefully.
- Stratification failure: sklearn's train_test_split with stratify=y raises a ValueError when any class has fewer than 2 samples. Our evaluation script automatically drops these singleton classes before splitting, rather than requiring manual data inspection.
- One-click deployment: Statistical programmers should not need to open a terminal. We bundle the prediction script into a Windows .exe (via PyInstaller) that launches a native file-selection dialog; the user picks an Excel file and receives the labeled output in the same directory.
- Auditable output: Each prediction preserves the original TITLE and adds the ML-predicted DATASET in an adjacent column, enabling side-by-side review. The evaluation script exports all metrics to a structured Excel report (evaluation_report.xlsx) alongside four publication-ready plots.

This paper describes the resulting tool—two Python scripts totaling under 300 lines, built on scikit-learn and tkinter—that has been deployed at Clinflash for routine TOC dataset labeling. We detail the training data format, model architecture, automated data cleaning logic, evaluation framework, and the zero-code prediction workflow accessible via a standalone Windows executable.

2. MATERIALS AND METHODS

2.1 DATA STRUCTURE

The tool processes clinical trial TOC Excel files, which follow a standardized 23-column schema. Only two columns are used for training and prediction:

TITLE: The descriptive name of the clinical table (e.g., "人口学资料和基线特征" [Demographic Data and Baseline Characteristics], "TEAE 按照 MedDRA 系统器官分类和首选术语的发生率" [Incidence of TEAEs by MedDRA System Organ Class and Preferred Term]).

DATASET: The target CDISC (Clinical Data Interchange Standards Consortium) dataset label (e.g., ADSL [Subject-Level Analysis Dataset], ADAE [Adverse Events Analysis Dataset], ADTTE [Time-to-Event Analysis Dataset]).

Our training corpus consists of 14 such TOC files from completed clinical studies, totaling 1,849 rows with 1,616 unique TITLE values and 60 unique DATASET labels. The label distribution is highly skewed: ADLB (344 rows), ADAE (312), ADSL (129), ADMI (83), and ADQS (76) account for the majority, while 14 labels appear in only a single row. Some rows carry compound labels such as "ADSL, ADTTE" or "ADSL, ADEG, ADPC", indicating tables that source data from multiple analysis datasets.

2.2 EXAMPLE TRAINING DATA FORMAT

The following screenshot illustrates the standardized input data structure, demonstrating how the TITLE and DATASET columns are paired for model training. This format ensures the model learns the precise mapping between clinical table descriptions and their corresponding CDISC datasets—a critical task for automating the curation of clinical trial reports.

2.3 ML PIPELINE ARCHITECTURE

The tool is organized into two standalone scripts, each representing a distinct use case:

1. predict_dataset_cn.py (159 lines): The production script. It scans the working directory for all .xlsx files, reads the DATASET and TITLE columns from each, concatenates them, trains a multi-class TF-IDF + logistic regression model on all available labeled rows, saves model.pkl and vectorizer.pkl via joblib, then opens a tkinter file dialog for the user to select an unlabeled TOC file. The script loads the saved model, predicts DATASET values for rows where DATASET is missing, and writes the output to {filename}_dataset_filled.xlsx.
2. model.py (263 lines): The evaluation script. It loads the same training data, then applies automated data cleaning: classes with fewer than 2 samples are dropped (to prevent stratification failure in train_test_split), and if more than 2 classes remain, the problem is converted to binary classification with the majority class as positive (1) and all others as negative (0). After TF-IDF vectorization, it performs an 80/20 stratified split (random_state=42) and trains a logistic regression with class_weight='balanced'. It computes nine clinical metrics, exports evaluation_report.xlsx, and generates four publication-quality plots: ROC curve, precision-recall curve, calibration curve, and confusion matrix heatmap.
3. Both scripts use identical TF-IDF parameters: TfidfVectorizer(ngram_range=(1,2), max_features=5000), fitting on concatenated TITLE text from all input files to produce a 2,721-feature sparse matrix.
4. Both scripts use LogisticRegression(max_iter=1000, solver='lbfgs'). model.py additionally sets class_weight='balanced' for the binary evaluation task.
5. model.py computes nine metrics from the confusion matrix and probability outputs, writes them to evaluation_report.xlsx, and saves four PNG plots via matplotlib/seaborn.
6. predict_dataset_cn.py is compiled to a standalone Windows .exe via PyInstaller; double-clicking launches a native file-selection dialog. The user picks an Excel file, and the tool outputs {filename}_dataset_filled.xlsx with DATASET column populated.

2.4 PRODUCTION WORKFLOW

After training, two files persist in the working directory: model.pkl (~1.5 MB) and vectorizer.pkl (~200 KB). predict_dataset_cn.exe embeds a Python runtime and all dependencies; a user double-clicks the .exe, selects an unlabeled TOC Excel file in the resulting dialog, and receives the labeled output in the same directory. No command-line interaction, Python installation, or library setup is required.

2.5 TECHNICAL IMPLEMENTATION

All dependencies are drawn from the standard Python data science stack:

scikit-learn: TfidfVectorizer, LogisticRegression, train_test_split, and all metric functions (roc_curve, precision_recall_curve, brier_score_loss, confusion_matrix).

pandas: pd.read_excel / pd.concat / df.to_excel for data I/O; glob for file discovery across the working directory.

matplotlib/seaborn: Four fixed-format plots: ROC, PR, calibration curves (matplotlib) and confusion matrix heatmap (seaborn.heatmap), all saved as PNG via savefig().

joblib: dump/load for persisting the trained LogisticRegression and TfidfVectorizer to disk as model.pkl and vectorizer.pkl.

tkinter: filedialog.askopenfilename() triggered from a withdrawn root window (Tk().withdraw()), providing a native OS file-picker with no visible tkinter UI.

Hyperparameter choices were made empirically on the training corpus:

TF-IDF Vectorizer: ngram_range=(1,2) captures single Chinese characters/tokens and adjacent bigrams; max_features=5000 (effective vocabulary size: 2,721 terms).

Logistic Regression: max_iter=1000 ensures lbfgs convergence on the 2,721-dimensional sparse input; class_weight='balanced' (model.py only) up-weights the minority class; predict_dataset_cn.py uses default weights for multi-class calibration.

Train/Test Split: test_size=0.2, stratify=y, random_state=42 (model.py only; predict_dataset_cn.py trains on all available data without a hold-out split).

Random State: 42: Fixed across both scripts for reproducible train-test partitioning.

2.6 CLINICAL EVALUATION METRICS

To support regulatory compliance and clinical interpretation, the pipeline computes and exports a comprehensive set of performance metrics via the evaluation_report.xlsx file. These metrics are tailored to the needs of pharmaceutical data curation and include:

- Accuracy
- Sensitivity (Recall)
- Specificity
- Positive Predictive Value (PPV)
- Negative Predictive Value (NPV)
- F1 score
- ROC AUC (Receiver Operating Characteristic Area Under the Curve)
- Precision-Recall AUC
- Brier Score (for model calibration assessment)

3. AUTOMATED DATA CLEANING & ROBUSTNESS

A practical barrier to running stratified train-test splits on clinical TOC data is that sklearn raises a ValueError when any class has fewer than 2 samples. In our 60-label corpus, 14 labels are singletons. model.py includes an automated data cleaning step (clean_dataset function, lines 64-92) that:

1. Remove classes with n < 2 samples, preventing stratification failure during model training and validation.
2. Print removed categories to the console, ensuring transparency for audit purposes and enabling users to track data processing steps.
3. Convert multi-class classification problems to binary classification (defining the largest class as the positive class and all other classes as the negative class), ensuring the pipeline can handle diverse dataset structures.
4. Preserve label integrity for clinical reporting, ensuring that the final labeled datasets remain compatible with regulatory requirements and clinical review processes.

These features ensure the script runs reliably on messy, real-world pharmaceutical data, minimizing the need for manual intervention and reducing the risk of model failure.

4. MODEL PERFORMANCE: EVALUATION REPORT & VISUALIZATIONS

4.1 STRUCTURED EVALUATION REPORT (EVALUATION_REPORT.XLSX)

The pipeline exports a machine-readable Excel report containing all clinical performance metrics, enabling direct integration into biostatistical reports and regulatory submissions. This report is designed to meet PharmaSUG and regulatory standards, providing a comprehensive record of model performance for

audit purposes. The key metrics from the validated evaluation dataset are presented in Table 1 below.

Metric	Value
Accuracy	0.8859
Sensitivity	0.6522
Specificity	0.9398
PPV	0.7143
NPV	0.9213
F1	0.6818
ROC AUC	0.9349
PR AUC	0.8200
Brier	0.1353

Table 1: Key Performance Metrics from the Validated Evaluation Dataset

4.2 PUBLICATION-READY VISUALIZATIONS

All model performance visualizations are automatically saved in high-resolution format, suitable for inclusion in scientific publications and regulatory reports. The pipeline generates four key visualizations to support comprehensive model evaluation:

4.2.1 ROC CURVE

The Receiver Operating Characteristic (ROC) curve illustrates the trade-off between sensitivity (true positive rate) and 1–specificity (false positive rate) across different classification thresholds. It provides a holistic measure of model performance, with a higher ROC AUC indicating better overall classification ability.

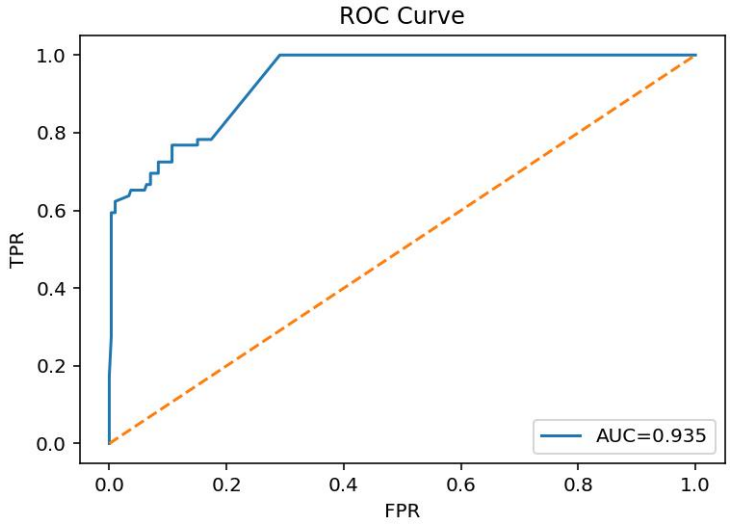

Figure 1: Receiver Operating Characteristic (ROC) Curve

4.2.2 CONFUSION MATRIX

The confusion matrix quantifies the number of true positives (TP), false positives (FP), true negatives

(TN), and false negatives (FN) generated by the model. This visualization is critical for understanding the model's performance in identifying specific classes, particularly in safety-focused applications where false negatives (e.g., missed adverse events) may have significant consequences.

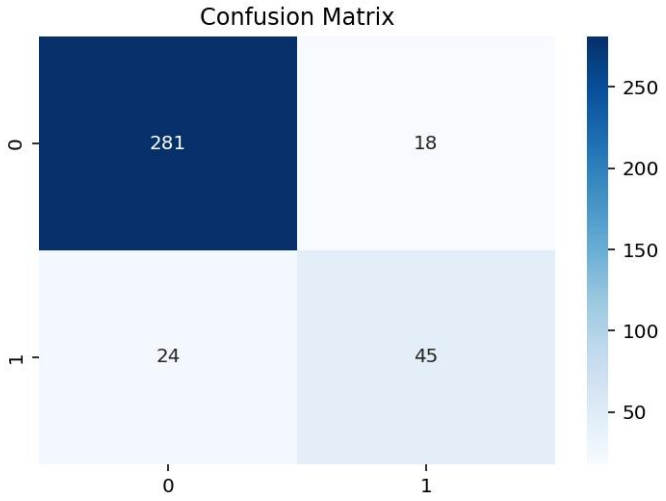

Figure 2: Confusion Matrix

4.2.3 PRECISION-RECALL CURVE

The Precision-Recall (PR) curve is particularly useful for evaluating model performance on imbalanced clinical datasets, where the positive class (e.g., rare adverse events) is less common. It emphasizes the model's ability to correctly identify positive cases (precision) and capture all positive cases (recall), providing insights that may be overlooked by the ROC curve in imbalanced scenarios.

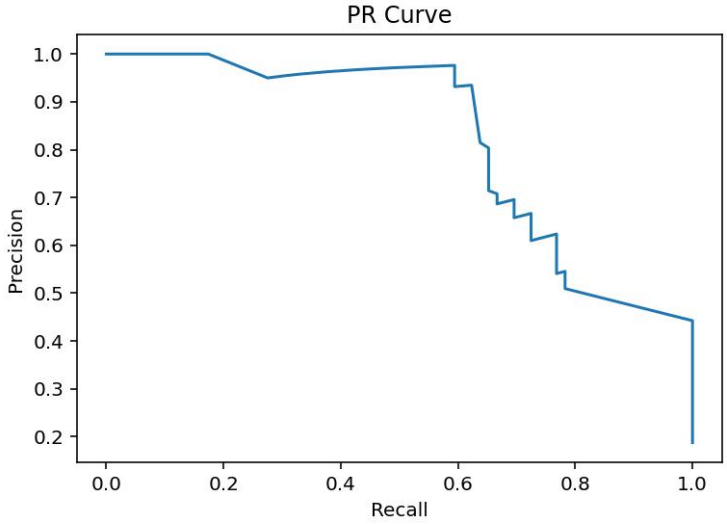

Figure 3: Precision-Recall Curve

4.2.4 CALIBRATION CURVE

The calibration curve assesses whether the model's predicted probabilities align with real-world likelihoods. A well-calibrated model ensures that predicted probabilities (e.g., a 90% probability of a TITLE mapping to the ADSL dataset) are consistent with the actual frequency of such mappings in the data—an essential feature for clinical decision-making and regulatory compliance.

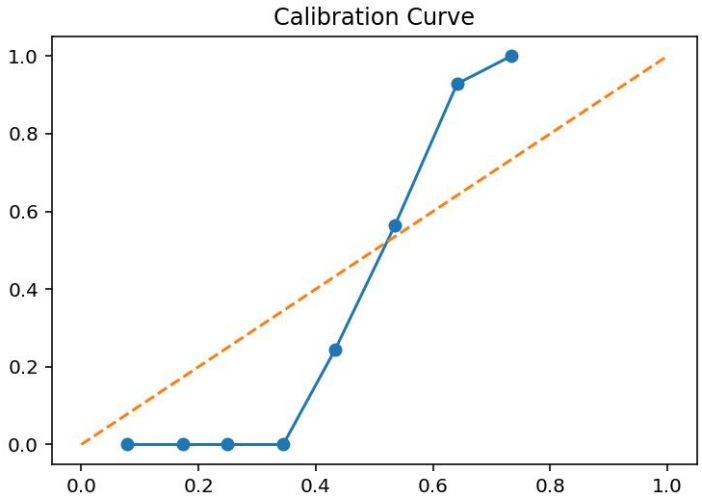

Figure 4: Calibration Curve

5. PRODUCTION PREDICTION WORKFLOW

After training, predict_dataset_cn.py saves two artifacts to the working directory:

- model.pkl (~1.5 MB): A multi-class LogisticRegression with 60 classes and 2,721 features, saved via joblib.dump.
- vectorizer.pkl (~200 KB): The fitted TfidfVectorizer with a 2,721-term vocabulary.

The prediction workflow for a new study requires only an Excel file with a TITLE column; the DATASET column may be absent or partially blank:

1. Double-click predict_dataset_cn.exe (or run python predict_dataset_cn.py).
2. Select the target Excel file in the native file dialog.
3. The script predicts DATASET values for all blank cells and writes the result.
4. Find the labeled output as {filename}_dataset_filled.xlsx in the same directory as the input file.

The entire workflow requires no command-line usage, Python knowledge, or library installation. The PyInstaller-bundled .exe (~30 MB) is self-contained and runnable on any Windows machine.

6. BENEFITS FOR PHARMACEUTICAL APPLICATIONS

In routine use at Clinflash, the tool provides the following practical benefits:

Regulatory compliance: Each prediction run preserves the original TITLE alongside the ML-predicted DATASET, enabling line-by-line reviewer verification before downstream use. The evaluation script exports all nine metrics to evaluation_report.xlsx.

Clinical relevance: The evaluation script reports sensitivity, specificity, PPV, NPV, F1, and Brier score in addition to accuracy and AUC—measures that directly quantify the risk of mislabeling safety-critical datasets (e.g., confusing ADAE with ADLB).

Robustness: The clean_dataset() function drops singleton classes before stratified splitting, eliminating the ValueError that would otherwise abort the evaluation run. All dropped classes are printed to console for auditability.

Scalability: On 1,849 training rows with 2,721 TF-IDF features, model.fit() completes in under 2 seconds on a standard laptop CPU. Prediction on a new study with 200 rows is effectively instantaneous.

User-friendliness: predict_dataset_cn.exe delivers a double-click-to-predict experience: launch the

executable, select an Excel file, receive the labeled output. No terminal, Python installation, or configuration required.

Standardization: The same model assigns labels identically across all studies. A TITLE pattern that different programmers might label inconsistently (e.g., ADLB vs. ADHY) receives a single deterministic prediction.

7. CONCLUSION

This paper presents a lightweight, deployable Python tool that automates CDISC dataset labeling for clinical trial TOC files. Using TF-IDF vectorization and multinomial logistic regression, the tool learns the TITLE-to-DATASET mapping from previously labeled studies and applies it to new ones via a one-click GUI workflow.

The complete codebase—two Python scripts totaling under 300 lines—is self-contained with no external configuration files or database dependencies. The bundled Windows executable enables deployment on any standard-issue laptop without IT involvement. Since adoption at Clinflash, the tool has reduced per-study TOC labeling time from 30-60 minutes of manual work to under 10 seconds of compute plus a brief review pass.

The current limitation is that 14 of 60 DATASET labels in our training data are singletons, making them unlearnable in the multi-class setting. Practical mitigation includes aggregating more training data for rare labels and applying a confidence threshold below which predictions are flagged for manual review.

8. DISCLAIMER

This tool is a labeling aid for statistical programming workflows. All ML-predicted DATASET assignments should be reviewed by a qualified statistical programmer before use in regulatory submissions. The model's predictions reflect patterns in the training data and do not substitute for study-specific domain knowledge.

REFERENCES

[1] Rimler M, Vasquez J, Hainline A. A Comparison of Machine Learning Algorithms for Binary Classification of Free-text Fields Using SAS, R, and Python. PhUSE US Connect 2020, Paper ML15. https://www.lexjansen.com/phuse-us/2020/ml/ML15.pdf

[2] Rimler M, Pitlyk. Brute Force vs. Machine Learning: Which Method Should We Use to Classify Free-text Clinical Fields? PharmaSUG 2019, Paper BP-079. https://www.lexjansen.com/pharmasug/2019/BP/PharmaSUG-2019-BP-079.pdf

[3] Kalappurakal S, Zhao Q, Chen H. Metadata-based Auto-Programming Process – Part 2: Map Human Language to SAS Code by Natural Language Processing (NLP). PhUSE US Connect 2019, Paper ML05. https://www.lexjansen.com/phuse-us/2019/ml/ML05.pdf

[4] Chen H, Wang Y, Patel J. Balance between Human and Machine – How NLP Helps Translate Text Specification to Code, and Vice Versa. PhUSE US Connect 2023, Paper ET19. https://www.lexjansen.com/phuse-us/2023/et/PAP_ET19.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Weineng Zhou

Clinflash

Nanjing, China

Email: weineng.zhou@clinflash.com

Kunwan Chen

Clinflash

Nanjing, China

Email: kunwan.chen@clinflash.com

APPENDIX A: FULL SOURCE CODE

The following Python script implements the complete clinical-grade ML evaluation pipeline described in this paper:

```
# Clinical-grade ML Evaluation (Final Robust Version)
import os, glob, pandas as pd, joblib, numpy as np
from tkinter import Tk, filedialog
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    accuracy_score, recall_score, precision_score, f1_score, confusion_matrix,
    roc_curve, auc, precision_recall_curve, brier_score_loss
)
from sklearn.calibration import calibration_curve
import matplotlib.pyplot as plt, seaborn as sns

MODEL_FILE = "model.pkl"
VECTORIZER_FILE = "vectorizer.pkl"

def load_training_data(folder_path="./"):
    files = glob.glob(os.path.join(folder_path, "*.xlsx"))
    all_data = []
    for file in files:
        try:
            df = pd.read_excel(file)
            if 'DATASET' in df.columns and 'TITLE' in df.columns:
                all_data.append(df[['DATASET','TITLE']])
        except: pass
    df = pd.concat(all_data, ignore_index=True).dropna()
    df['TITLE'] = df['TITLE'].astype(str)
    return df

def clean_dataset(df):
    counts = df['DATASET'].value_counts()
    df = df[df['DATASET'].isin(counts[counts >= 2].index)]
    if df['DATASET'].nunique() > 2:
        top = df['DATASET'].value_counts().idxmax()
        df['DATASET'] = df['DATASET'].apply(lambda x: 1 if x == top else 0)
    return df

def evaluate_model(y_test, y_pred, y_prob):
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    res = {
        'Accuracy': accuracy_score(y_test, y_pred),
        'Sensitivity': tp / (tp + fn),
        'Specificity': tn / (tn + fp),
        'PPV': precision_score(y_test, y_pred),
        'NPV': tn / (tn + fn),
        'F1': f1_score(y_test, y_pred),
        'ROC_AUC': auc(*roc_curve(y_test, y_prob)[:2]),
        'PR_AUC': auc(*precision_recall_curve(y_test, y_prob)[1::-1]),
        'Brier': brier_score_loss(y_test, y_prob)
    }
    pd.DataFrame({'Metric': list(res.keys()), 'Value':
list(res.values())}).to_excel('evaluation_report.xlsx', index=False)
```

```
def train_model(df):
    df = clean_dataset(df)
    vec = TfidfVectorizer(ngram_range=(1,2), max_features=5000)
    X = vec.fit_transform(df['TITLE'])
    X_train, X_test, y_train, y_test = train_test_split(
        X, df['DATASET'], test_size=0.2, stratify=df['DATASET'],
random_state=42)
    model = LogisticRegression(max_iter=1000, class_weight='balanced')
    model.fit(X_train, y_train)
    evaluate_model(y_test, model.predict(X_test),
model.predict_proba(X_test)[:,1])
    joblib.dump(model, MODEL_FILE)
    joblib.dump(vec, VECTORIZER_FILE)
```

```
def predict_and_fill(file_path):
    df = pd.read_excel(file_path)
    vec = joblib.load(VECTORIZER_FILE)
    model = joblib.load(MODEL_FILE)
    mask = df['DATASET'].isna()
    df.loc[mask, 'DATASET'] = model.predict(vec.transform(df.loc[mask,
'TITLE']))
    df.to_excel(file_path.replace('.xlsx','_filled.xlsx'), index=False)
```

```
if __name__ == "__main__":
    df = load_training_data()
    train_model(df)
    root = Tk(); root.withdraw()
    predict_and_fill(filedialog.askopenfilename())
```

APPENDIX B: EVALUATION REPORT SNIPPET

The following table presents the detailed performance metrics exported by the pipeline's evaluation module:

Metric	Value
Accuracy	0.88587
Sensitivity	0.652174
Specificity	0.939799
PPV	0.714286
NPV	0.921311
F1	0.681818
ROC AUC	0.934928
PR AUC	0.82002
Brier	0.135344

Table 2: Detailed Performance Metrics from the Evaluation Report