

NLP Assignment (Imbalanced Dataset)

Subkitted

This notebook:

- Mounts Google Drive
- Creates a project folder named **NLP assignmet**
- Creates subfolders (including **data/** for your dataset)
- Provides a clean starting pipeline (preprocess → split → TF-IDF → baseline RF → over/under/SMOTE)

Put your dataset file (e.g., `unbalanceddataset.csv`) into:
MyDrive/NLP assignmet/data/

```
In [15]: # =====
# 1) Install + Imports (FULL)
# =====
!pip -q install pandas numpy scikit-learn nltk imbalanced-learn matplotlib

# ---- Core ----
import re
import numpy as np
import pandas as pd
from pathlib import Path

# ---- NLTK ----
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer

# NLTK resources (including punkt_tab fix)
nltk.download('punkt')
nltk.download('punkt_tab')      # ✅ required by some NLTK versions for to
nltk.download('stopwords')
nltk.download('wordnet')

# ---- scikit-learn: split + features ----
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectori

# ---- scikit-learn: models ----
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

# ---- scikit-learn: preprocessing (optional) ----
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```
# ---- Metrics ----
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, classification_report
)

# ---- Imbalanced-learn ----
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler

# ---- Visualization ----
import matplotlib.pyplot as plt
import seaborn as sns

# ---- spaCy (optional in lecture, included here) ----
import spacy
try:
    _ = spacy.load("en_core_web_sm")
except OSError:
    !python -m spacy download en_core_web_sm
    _ = spacy.load("en_core_web_sm")

print("✅ All libraries imported successfully (NLTK + sklearn + imblearn
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data] Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Package wordnet is already up-to-date!
```

```
✅ All libraries imported successfully (NLTK + sklearn + imblearn + plotti
ng + spaCy)
```

```
In [16]: # =====
# 2) Mount Google Drive
# =====
from google.colab import drive
from pathlib import Path
import os
import datetime

drive.mount('/content/drive', force_remount=False)

# =====
# 3) Create project folders
# =====
PROJECT_NAME = "NLP assignmet"
BASE_DIR = Path("/content/drive/MyDrive") / PROJECT_NAME

DIRS = {
    "data": BASE_DIR / "data",
    "notebooks": BASE_DIR / "notebooks",
    "src": BASE_DIR / "src",
    "models": BASE_DIR / "models",
    "outputs": BASE_DIR / "outputs",
    "figures": BASE_DIR / "figures",
    "reports": BASE_DIR / "reports",
    "logs": BASE_DIR / "logs",
```

```

}

for p in DIRS.values():
    p.mkdir(parents=True, exist_ok=True)

run_stamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
run_dir = DIRS["outputs"] / f"run_{run_stamp}"
run_dir.mkdir(parents=True, exist_ok=True)

os.environ["NLP_PROJECT_DIR"] = str(BASE_DIR)
os.environ["NLP_RUN_DIR"] = str(run_dir)

print("✅ Project folders created at:")
print(BASE_DIR)
print("\n📁 Folder structure:")
for key in sorted(DIRS.keys()):
    print(f"  {key:10s}: {DIRS[key]}")
print("\n📁 Run output folder:")
print(run_dir)

```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

✅ Project folders created at:
/content/drive/MyDrive/NLP assignmet

📁 Folder structure:

- data : /content/drive/MyDrive/NLP assignmet/data
- figures : /content/drive/MyDrive/NLP assignmet/figures
- logs : /content/drive/MyDrive/NLP assignmet/logs
- models : /content/drive/MyDrive/NLP assignmet/models
- notebooks : /content/drive/MyDrive/NLP assignmet/notebooks
- outputs : /content/drive/MyDrive/NLP assignmet/outputs
- reports : /content/drive/MyDrive/NLP assignmet/reports
- src : /content/drive/MyDrive/NLP assignmet/src

📁 Run output folder:

/content/drive/MyDrive/NLP assignmet/outputs/run_20251213_071709

Importing the dataset in the data folder

- I have uploaded your CSV (example name: `unbalanceddataset.csv`) to:
 - `MyDrive/NLP assignmet/data/`

```

In [6]: # =====
# 4) Locate dataset file
# =====
DATA_FILE = DIRS['data'] / 'unbalanceddataset.csv' # change if your fil

if DATA_FILE.exists():
    print('✅ Found dataset:', DATA_FILE)
else:
    print('⚠️ Dataset not found at:', DATA_FILE)
    print('👉 Upload your CSV to:', DIRS['data'])
    print('\nCurrent files in data folder:')
    for f in DIRS['data'].glob('*'):
        print(' -', f.name)

```

✅ Found dataset: /content/drive/MyDrive/NLP assignmet/data/unbalanceddataset.csv

Pipeline (preprocess → split → TF-IDF → RF → resampling)

If your file is present, run the cells below.

```
In [7]: # =====
# 5) Load data
# =====
import pandas as pd

df = pd.read_csv(DATA_FILE)
print('Shape:', df.shape)
print('Columns:', list(df.columns))
print('\nClass distribution:')
print(df['sentiment'].value_counts())
df.head()
```

```
Shape: (2600, 2)
Columns: ['text', 'sentiment']
```

```
Class distribution:
sentiment
positive    2000
negative     600
Name: count, dtype: int64
```

```
Out[7]:
```

	text	sentiment
0	Java Concurrency in Practice is probably the b...	positive
1	haha aww hun i bet you are more creative tha...	positive
2	_pickle lol, thank you very much Hope you`re h...	positive
3	Out for an evening on the town with jeremy. Sa...	negative
4	- just took over the #1 Most Endorsed spot on...	positive

```
In [18]: # =====
# 6) Text preprocessing + tables + plots
# =====
import re
import os
from typing import Optional
import pandas as pd
import matplotlib.pyplot as plt

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer

stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words("english"))

_URL_RE = re.compile(r"http\S+|www\.\S+", flags=re.IGNORECASE)
```

```

_HTML_RE = re.compile(r"<.*?>")
_NON_ALPHA_RE = re.compile(r"^[a-zA-Z\s']+")
_WS_RE = re.compile(r"\s+")

def preprocess_text(text: Optional[str], use_lemmatization: bool = False)
    if text is None:
        return ""
    text = str(text).lower().strip()
    if not text:
        return ""
    text = _URL_RE.sub(" ", text)
    text = _HTML_RE.sub(" ", text)
    text = _NON_ALPHA_RE.sub(" ", text)
    text = _WS_RE.sub(" ", text).strip()
    if not text:
        return ""

    tokens = word_tokenize(text)
    tokens = [t for t in tokens if t not in stop_words and len(t) > 1]

    if use_lemmatization:
        tokens = [lemmatizer.lemmatize(t) for t in tokens]
    else:
        tokens = [stemmer.stem(t) for t in tokens]

    return " ".join(tokens)

required_cols = {"text", "sentiment"}
missing = required_cols - set(df.columns)
if missing:
    raise ValueError(f"Missing required columns: {missing}")

raw_rows = len(df)
raw_counts = df["sentiment"].value_counts(dropna=False)

df = df.copy()
df["text"] = df["text"].astype(str)
df["clean_text"] = df["text"].apply(lambda s: preprocess_text(s, use_lemm
df["clean_text"] = df["clean_text"].fillna("").astype(str).str.strip()

df_clean = df.dropna(subset=["sentiment"]).copy()
df_clean = df_clean[df_clean["clean_text"].str.len() > 0].reset_index(drop

clean_rows = len(df_clean)
clean_counts = df_clean["sentiment"].value_counts(dropna=False)

summary_table = (
    pd.DataFrame({"Before": raw_counts, "After": clean_counts})
    .fillna(0)
    .astype(int)
    .sort_index()
)

dropped_rows = raw_rows - clean_rows
dropped_pct = (dropped_rows / raw_rows * 100) if raw_rows else 0.0

print("✅ Preprocessing done.")
print(f"Rows before: {raw_rows} | Rows after: {clean_rows} | Dropped: {dr
print("\nClass distribution (Before vs After):")
display(summary_table)

```

```

if clean_rows > 0:
    print("\nExample:")
    print("RAW :", df_clean.loc[0, "text"][:120], "...")
    print("CLEAN:", df_clean.loc[0, "clean_text"][:120], "...")

ax = summary_table.plot(kind="bar")
ax.set_title("Class Distribution: Before vs After Preprocessing")
ax.set_xlabel("Class")
ax.set_ylabel("Count")
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()

fig_dir = None
if "DIRS" in globals() and isinstance(DIRS, dict) and "figures" in DIRS:
    fig_dir = DIRS["figures"]
elif os.getenv("NLP_PROJECT_DIR"):
    fig_dir = os.path.join(os.getenv("NLP_PROJECT_DIR"), "figures")

if fig_dir:
    os.makedirs(fig_dir, exist_ok=True)
    fig_path = os.path.join(str(fig_dir), "class_distribution_before_after.png")
    plt.figure()
    summary_table.plot(kind="bar")
    plt.title("Class Distribution: Before vs After Preprocessing")
    plt.xlabel("Class")
    plt.ylabel("Count")
    plt.xticks(rotation=0)
    plt.tight_layout()
    plt.savefig(fig_path, dpi=200)
    plt.close()
    print(f"Saved plot: {fig_path}")

X = df_clean["clean_text"]
y = df_clean["sentiment"]

```

✓ Preprocessing done.

Rows before: 2598 | Rows after: 2598 | Dropped: 0 (0.00%)

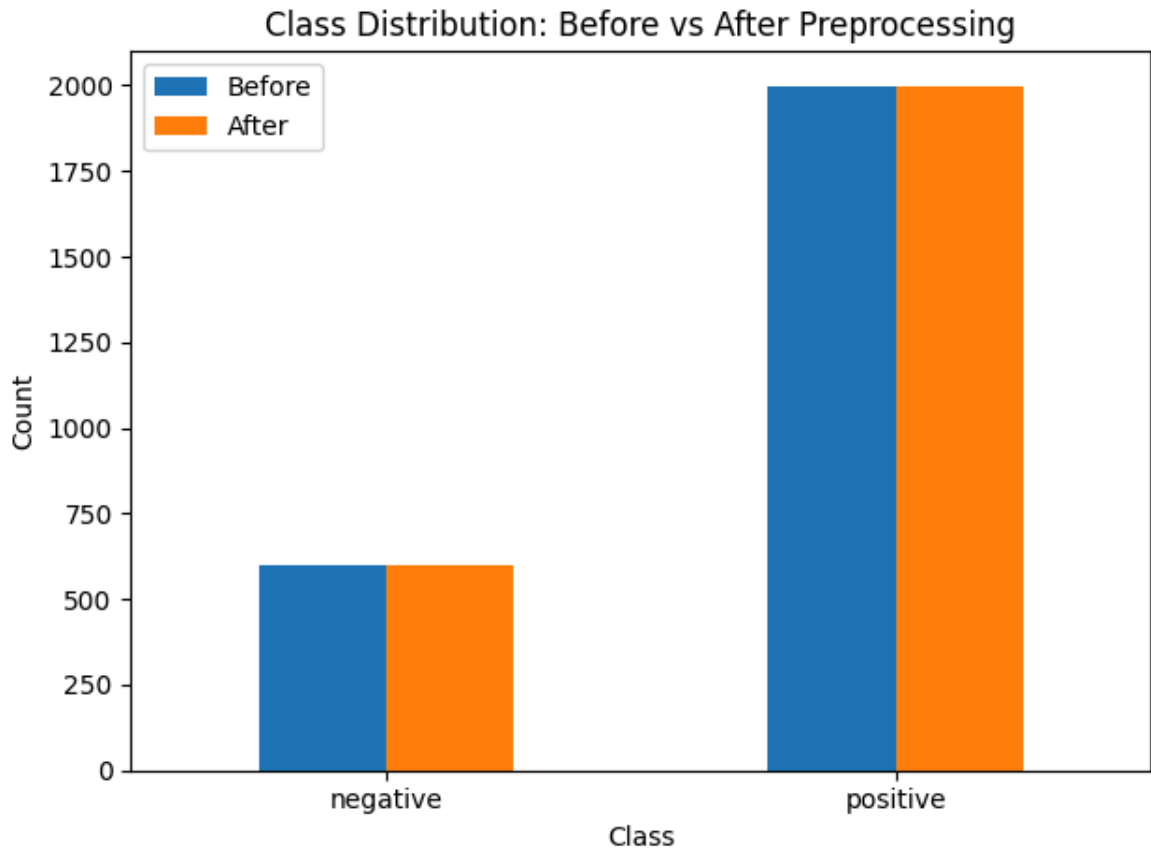
Class distribution (Before vs After):

	Before	After
sentiment		
negative	599	599
positive	1999	1999

Example:

RAW : Java Concurrency in Practice is probably the best Java book I've ever bought. There's a recipe in there for interrupting ...

CLEAN: java concurr practic probabl best java book ever bought recip interrupt block io op ...



Saved plot: /content/drive/MyDrive/NLP assignmet/figures/class_distribution_before_after.png

<Figure size 640x480 with 0 Axes>

```
In [19]: # =====
# 7) Split + TF-IDF (with tables + plots)
# =====
import os
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=42, stratify=y
)

split_table = pd.DataFrame(
    {
        "Train": y_train.value_counts(),
        "Test": y_test.value_counts(),
    }
).fillna(0).astype(int).sort_index()

print("✅ Split done.")
print(f"Train size: {len(X_train)} | Test size: {len(X_test)}")
print("\nClass distribution (Train vs Test):")
display(split_table)

ax = split_table.plot(kind="bar")
ax.set_title("Class Distribution: Train vs Test")
ax.set_xlabel("Class")
ax.set_ylabel("Count")
```

```

plt.xticks(rotation=0)
plt.tight_layout()
plt.show()

tfidf_config = {
    "max_features": 50000,
    "ngram_range": (1, 2),
    "min_df": 2,
    "max_df": 0.95,
    "sublinear_tf": True,
}

vectorizer = TfidfVectorizer(**tfidf_config)
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

print("\n✅ TF-IDF done.")
print("Vectorizer config:", tfidf_config)
print("Train shape:", X_train_vec.shape, "| Test shape:", X_test_vec.shape)

fig_dir = None
if "DIRS" in globals() and isinstance(DIRS, dict) and "figures" in DIRS:
    fig_dir = DIRS["figures"]
elif os.getenv("NLP_PROJECT_DIR"):
    fig_dir = os.path.join(os.getenv("NLP_PROJECT_DIR"), "figures")

if fig_dir:
    os.makedirs(fig_dir, exist_ok=True)
    fig_path = os.path.join(str(fig_dir), "class_distribution_train_test.")
    plt.figure()
    split_table.plot(kind="bar")
    plt.title("Class Distribution: Train vs Test")
    plt.xlabel("Class")
    plt.ylabel("Count")
    plt.xticks(rotation=0)
    plt.tight_layout()
    plt.savefig(fig_path, dpi=200)
    plt.close()
    print(f"Saved plot: {fig_path}")

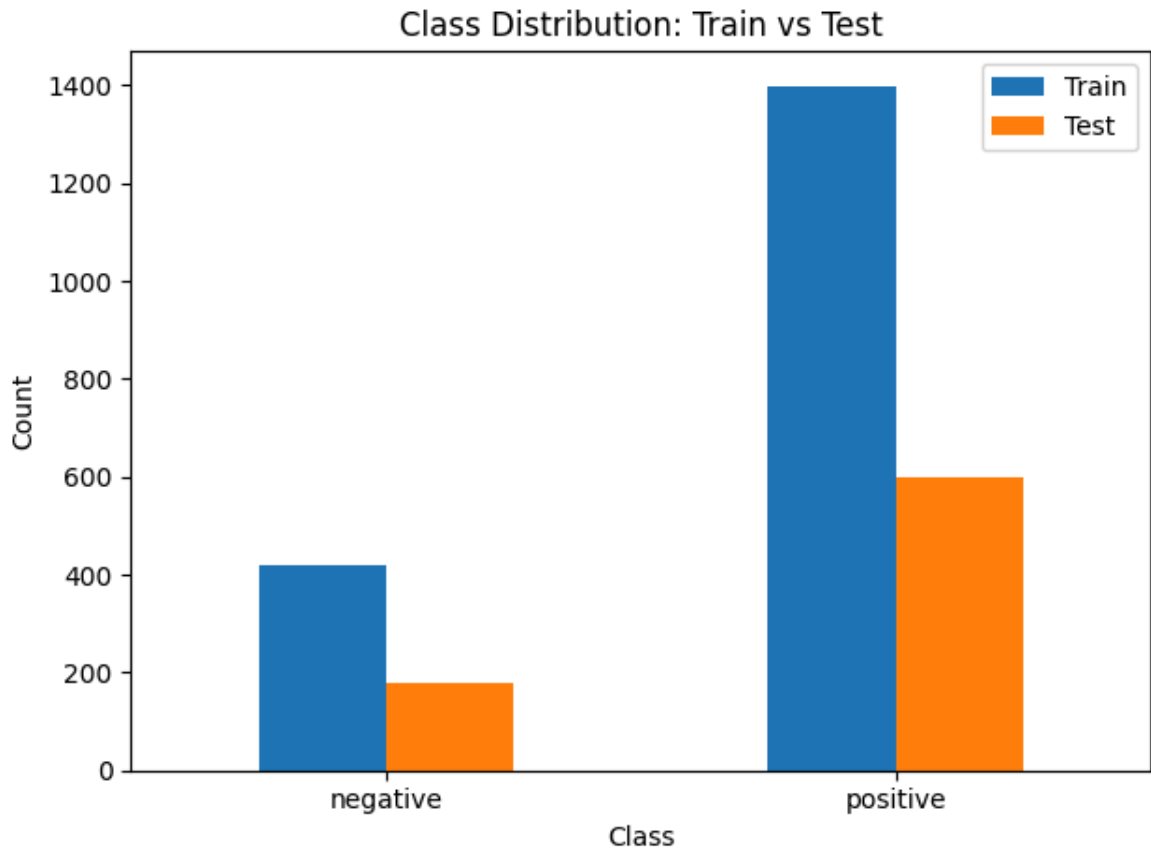
```

✅ Split done.

Train size: 1818 | Test size: 780

Class distribution (Train vs Test):

	Train	Test
sentiment		
negative	419	180
positive	1399	600



✅ TF-IDF done.

Vectorizer config: {'max_features': 50000, 'ngram_range': (1, 2), 'min_df': 2, 'max_df': 0.95, 'sublinear_tf': True}

Train shape: (1818, 1838) | Test shape: (780, 1838)

Saved plot: /content/drive/MyDrive/NLP assignmet/figures/class_distribution_train_test.png

<Figure size 640x480 with 0 Axes>

```
In [22]: # =====
# 8) Evaluation helper (tables + confusion matrix plot + saving)
# =====
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display

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

EVAL_RESULTS = []

def _get_output_dir():
    if "DIRS" in globals() and isinstance(DIRS, dict) and "outputs" in DIRS:
        return str(DIRS["outputs"])
    proj = os.getenv("NLP_PROJECT_DIR")
    if proj:
        return os.path.join(proj, "outputs")
```

```

    return None

def _get_fig_dir():
    if "DIRS" in globals() and isinstance(DIRS, dict) and "figures" in DIRS:
        return str(DIRS["figures"])
    proj = os.getenv("NLP_PROJECT_DIR")
    if proj:
        return os.path.join(proj, "figures")
    return None

def evaluate_model(model_name, y_true, y_pred, labels=None, save_artifacts=True):
    if labels is None:
        labels = sorted(pd.Series(y_true).dropna().unique().tolist())

    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred, average="weighted", zero_division=0)
    rec = recall_score(y_true, y_pred, average="weighted", zero_division=0)
    f1 = f1_score(y_true, y_pred, average="weighted", zero_division=0)

    cm = confusion_matrix(y_true, y_pred, labels=labels)
    report_df = pd.DataFrame(
        classification_report(y_true, y_pred, output_dict=True, zero_division=0)
    ).transpose()

    metrics_row = {
        "Model": model_name,
        "Accuracy": acc,
        "Precision_weighted": prec,
        "Recall_weighted": rec,
        "F1_weighted": f1,
    }
    EVAL_RESULTS.append(metrics_row)

    print("\n" + "=" * 70)
    print(model_name)

    display(pd.DataFrame([metrics_row]).set_index("Model"))

    print("\nConfusion Matrix:")
    display(pd.DataFrame(cm, index=labels, columns=labels))

    plt.figure()
    sns.heatmap(cm, annot=True, fmt="d", xticklabels=labels, yticklabels=labels)
    plt.title(f"Confusion Matrix - {model_name}")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.tight_layout()
    plt.show()

    print("\nClassification Report:")
    display(report_df)

    if save_artifacts:
        out_dir = _get_output_dir()
        fig_dir = _get_fig_dir()

        metrics_path = None
        report_path = None
        fig_path = None

```

```

    if out_dir:
        os.makedirs(out_dir, exist_ok=True)
        metrics_path = os.path.join(out_dir, "metrics_summary.csv")
        pd.DataFrame(EVAL_RESULTS).to_csv(metrics_path, index=False)

        safe_name = model_name.replace(" ", "_").replace("+", "plus")
        report_path = os.path.join(out_dir, f"classification_report_{safe_name}.csv")
        report_df.to_csv(report_path)

    if fig_dir:
        os.makedirs(fig_dir, exist_ok=True)
        safe_name = model_name.replace(" ", "_").replace("+", "plus")
        fig_path = os.path.join(fig_dir, f"confusion_matrix_{safe_name}.png")

        plt.figure()
        sns.heatmap(cm, annot=True, fmt="d", xticklabels=labels, yticklabels=labels)
        plt.title(f"Confusion Matrix - {model_name}")
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.tight_layout()
        plt.savefig(fig_path, dpi=200)
        plt.close()

    if metrics_path:
        print(f"Saved metrics to: {metrics_path}")
    if report_path:
        print(f"Saved report to: {report_path}")
    if fig_path:
        print(f"Saved confusion matrix to: {fig_path}")

    return metrics_row

def get_results_table():
    if not EVAL_RESULTS:
        return pd.DataFrame()
    return (
        pd.DataFrame(EVAL_RESULTS)
        .set_index("Model")
        .sort_values("F1_weighted", ascending=False)
    )

```

```

In [23]: # =====
# 9) Baseline: Random Forest + GridSearch (with evaluation + saving)
# =====
import time
import joblib

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

rf = RandomForestClassifier(random_state=42, n_jobs=-1)

param_grid = {
    "n_estimators": [100, 200, 300],
    "max_depth": [None, 10, 20, 30],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
}

grid = GridSearchCV(

```

```

    estimator=rf,
    param_grid=param_grid,
    scoring="f1_weighted",
    cv=5,
    n_jobs=-1,
    verbose=1,
)

t0 = time.time()
grid.fit(X_train_vec, y_train)
elapsed = time.time() - t0

best_rf = grid.best_estimator_
print(f"✅ Best params: {grid.best_params_}")
print(f"✅ CV best score (f1_weighted): {grid.best_score_:.4f}")
print(f"⌚ GridSearch time: {elapsed:.1f}s")

y_pred_base = best_rf.predict(X_test_vec)
evaluate_model("RandomForest (Baseline)", y_test, y_pred_base, save_artif

model_dir = None
if "DIRS" in globals() and isinstance(DIRS, dict) and "models" in DIRS:
    model_dir = str(DIRS["models"])
elif os.getenv("NLP_PROJECT_DIR"):
    model_dir = os.path.join(os.getenv("NLP_PROJECT_DIR"), "models")

if model_dir:
    os.makedirs(model_dir, exist_ok=True)
    joblib.dump(best_rf, os.path.join(model_dir, "rf_baseline_best.joblib")
    joblib.dump(vectorizer, os.path.join(model_dir, "tfidf_vectorizer.job
    print(f"Saved model to: {os.path.join(model_dir, 'rf_baseline_best.jo
    print(f"Saved vectorizer to: {os.path.join(model_dir, 'tfidf_vectoriz

```

Fitting 5 folds for each of 108 candidates, totalling 540 fits

✅ Best params: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}

✅ CV best score (f1_weighted): 0.8289

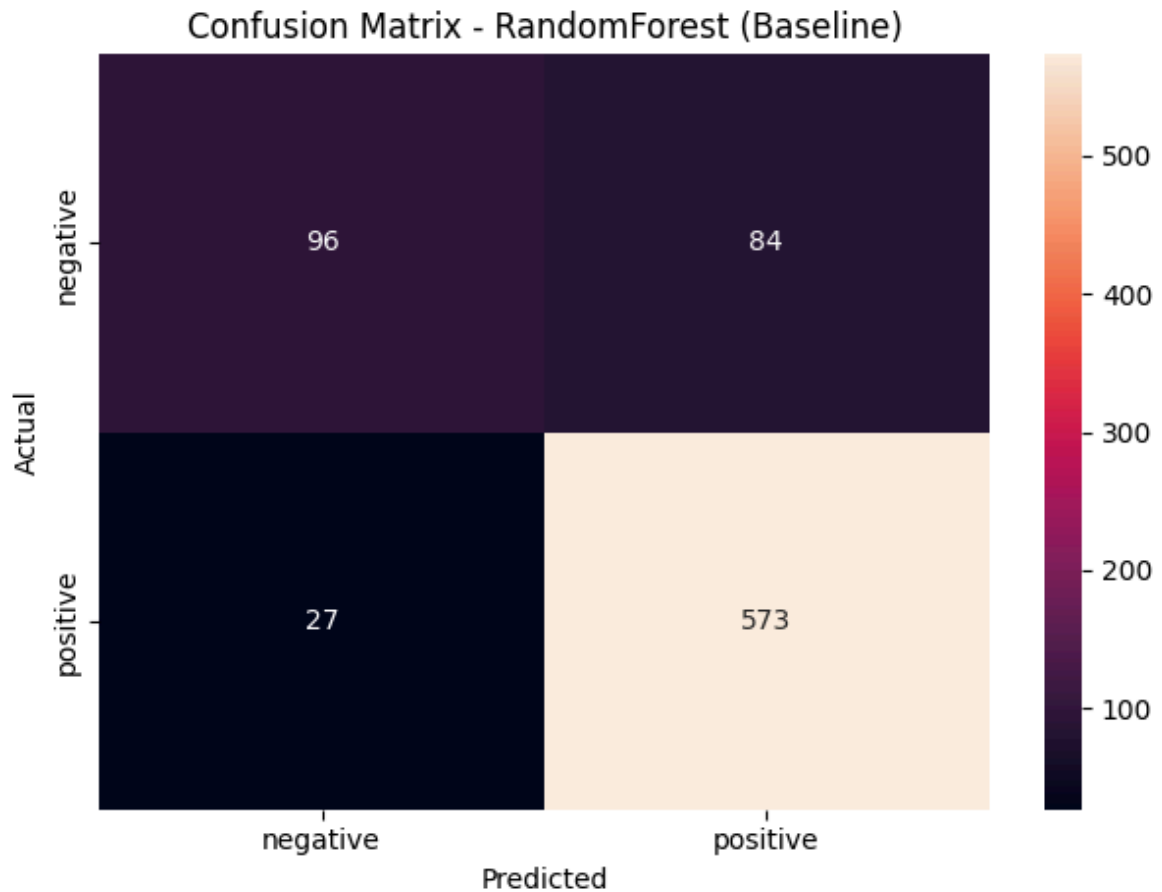
⌚ GridSearch time: 342.9s

=====
RandomForest (Baseline)

	Accuracy	Precision_weighted	Recall_weighted	F1_weighted
Model				
RandomForest (Baseline)	0.857692	0.850994	0.857692	0.847533

Confusion Matrix:

	negative	positive
negative	96	84
positive	27	573



Classification Report:

	precision	recall	f1-score	support
negative	0.780488	0.533333	0.633663	180.000000
positive	0.872146	0.955000	0.911695	600.000000
accuracy	0.857692	0.857692	0.857692	0.857692
macro avg	0.826317	0.744167	0.772679	780.000000
weighted avg	0.850994	0.857692	0.847533	780.000000

Saved metrics to: /content/drive/MyDrive/NLP assignmet/outputs/metrics_summary.csv

Saved report to: /content/drive/MyDrive/NLP assignmet/outputs/classification_report_RandomForest_(Baseline).csv

Saved confusion matrix to: /content/drive/MyDrive/NLP assignmet/figures/confusion_matrix_RandomForest_(Baseline).png

Saved model to: /content/drive/MyDrive/NLP assignmet/models/rf_baseline_best.joblib

Saved vectorizer to: /content/drive/MyDrive/NLP assignmet/models/tfidf_vectorizer.joblib

```
In [24]: # =====
# 10) Resampling experiments (Over / Under / SMOTE) + comparison table +
# =====
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from collections import Counter
```

```

from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

resamplers = [
    ("RF + RandomOverSampler", RandomOverSampler(random_state=42)),
    ("RF + RandomUnderSampler", RandomUnderSampler(random_state=42)),
    ("RF + SMOTE", SMOTE(random_state=42)),
]

fast_param_grid = {
    "n_estimators": [100, 200],
    "max_depth": [None, 20],
    "min_samples_split": [2, 5],
    "min_samples_leaf": [1, 2],
}

def _fit_rf_with_grid(X_res, y_res):
    rf = RandomForestClassifier(random_state=42, n_jobs=-1)
    grid = GridSearchCV(
        rf,
        param_grid=fast_param_grid,
        scoring="f1_weighted",
        cv=5,
        n_jobs=-1,
        verbose=0,
    )
    grid.fit(X_res, y_res)
    return grid

print("Train class distribution (before resampling):", Counter(y_train))

for model_name, sampler in resamplers:
    X_res, y_res = sampler.fit_resample(X_train_vec, y_train)
    print("\n" + "-" * 70)
    print(model_name)
    print("After resampling:", Counter(y_res))

    grid_rs = _fit_rf_with_grid(X_res, y_res)
    best_model = grid_rs.best_estimator_

    print("Best params:", grid_rs.best_params_)
    print(f"CV best score (f1_weighted): {grid_rs.best_score_:.4f}")

    y_pred = best_model.predict(X_test_vec)
    evaluate_model(model_name, y_test, y_pred, save_artifacts=True)

results_table = get_results_table()
print("\n✅ Final comparison table:")
display(results_table)

if not results_table.empty:
    ax = results_table["F1_weighted"].plot(kind="bar")
    ax.set_title("F1_weighted Comparison (Baseline vs Resampling)")
    ax.set_xlabel("Model")
    ax.set_ylabel("F1_weighted")
    plt.xticks(rotation=45, ha="right")
    plt.tight_layout()

```

```

plt.show()

fig_dir = None
if "DIRS" in globals() and isinstance(DIRS, dict) and "figures" in DIRS:
    fig_dir = str(DIRS["figures"])
elif os.getenv("NLP_PROJECT_DIR"):
    fig_dir = os.path.join(os.getenv("NLP_PROJECT_DIR"), "figures")

if fig_dir:
    os.makedirs(fig_dir, exist_ok=True)
    fig_path = os.path.join(fig_dir, "f1_comparison_baseline_vs_resampling")
    plt.figure()
    results_table["F1_weighted"].plot(kind="bar")
    plt.title("F1_weighted Comparison (Baseline vs Resampling)")
    plt.xlabel("Model")
    plt.ylabel("F1_weighted")
    plt.xticks(rotation=45, ha="right")
    plt.tight_layout()
    plt.savefig(fig_path, dpi=200)
    plt.close()
    print(f"Saved plot: {fig_path}")

```

Train class distribution (before resampling): Counter({'positive': 1399, 'negative': 419})

RF + RandomOverSampler

After resampling: Counter({'positive': 1399, 'negative': 1399})

Best params: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}

CV best score (f1_weighted): 0.9156

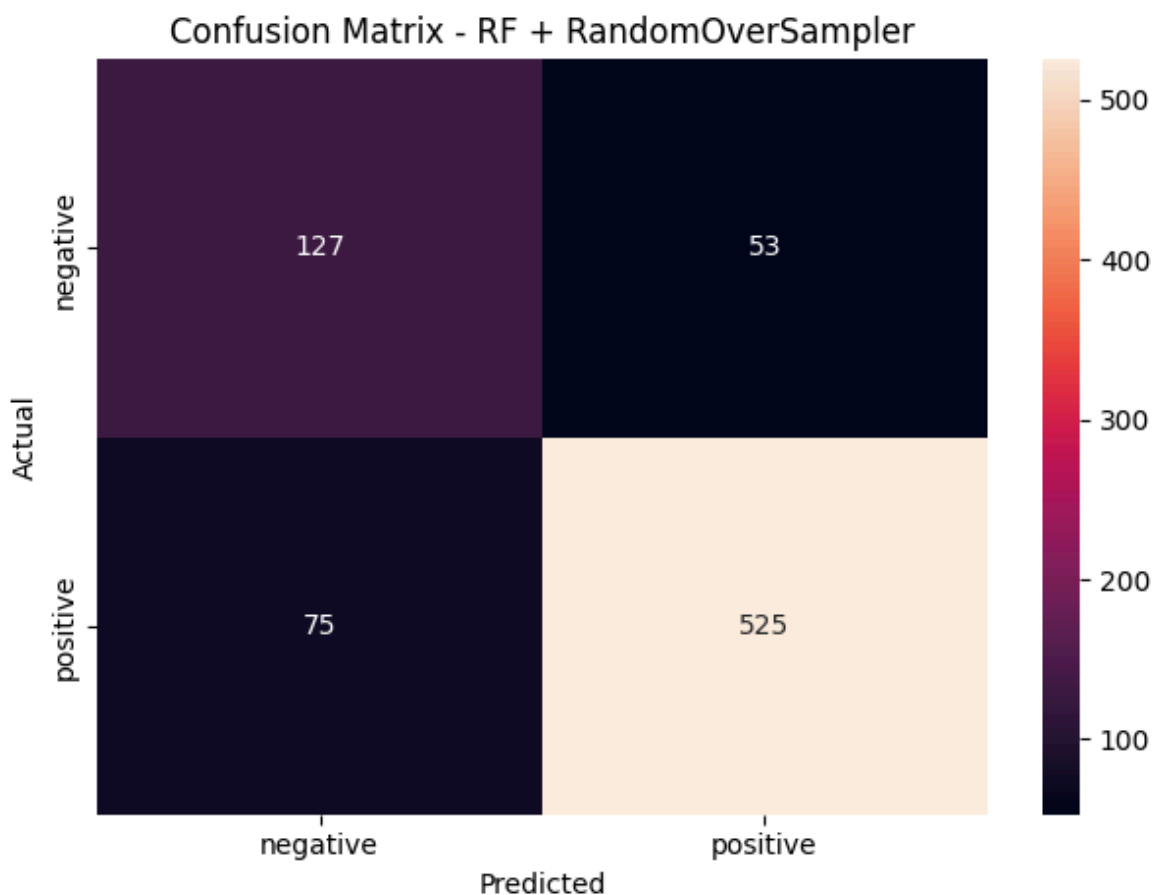
=====

RF + RandomOverSampler

	Accuracy	Precision_weighted	Recall_weighted	F1_weighted
Model				
RF + RandomOverSampler	0.835897	0.843783	0.835897	0.839091

Confusion Matrix:

	negative	positive
negative	127	53
positive	75	525



Classification Report:

	precision	recall	f1-score	support
negative	0.628713	0.705556	0.664921	180.000000
positive	0.908304	0.875000	0.891341	600.000000
accuracy	0.835897	0.835897	0.835897	0.835897
macro avg	0.768509	0.790278	0.778131	780.000000
weighted avg	0.843783	0.835897	0.839091	780.000000

Saved metrics to: /content/drive/MyDrive/NLP assignmet/outputs/metrics_summary.csv

Saved report to: /content/drive/MyDrive/NLP assignmet/outputs/classification_report_RF_plus_RandomOverSampler.csv

Saved confusion matrix to: /content/drive/MyDrive/NLP assignmet/figures/confusion_matrix_RF_plus_RandomOverSampler.png

=====

RF + RandomUnderSampler

After resampling: Counter({'negative': 419, 'positive': 419})

Best params: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}

CV best score (f1_weighted): 0.7985

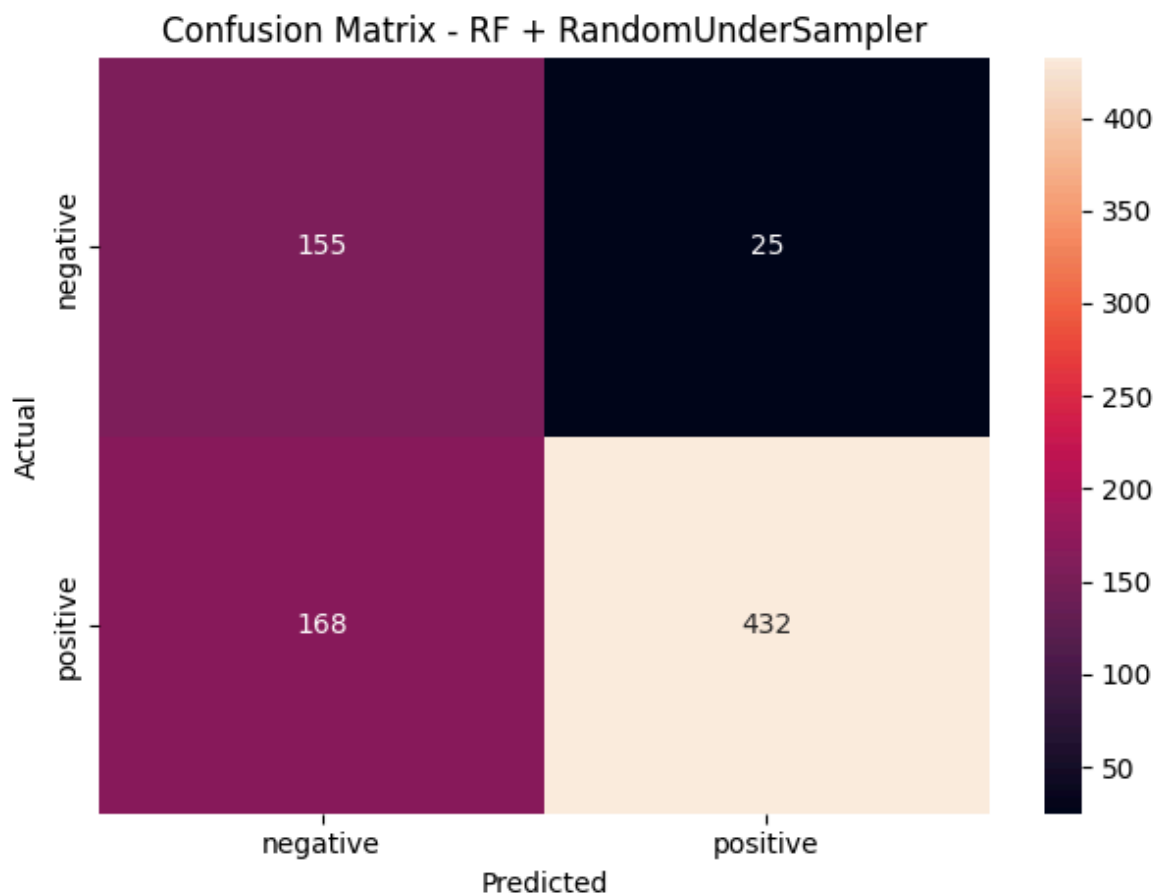
=====

RF + RandomUnderSampler

	Accuracy	Precision_weighted	Recall_weighted	F1_weighted
Model				
RF + RandomUnderSampler	0.752564	0.837891	0.752564	0.770999

Confusion Matrix:

	negative	positive
negative	155	25
positive	168	432



Classification Report:

	precision	recall	f1-score	support
negative	0.479876	0.861111	0.616302	180.000000
positive	0.945295	0.720000	0.817408	600.000000
accuracy	0.752564	0.752564	0.752564	0.752564
macro avg	0.712586	0.790556	0.716855	780.000000
weighted avg	0.837891	0.752564	0.770999	780.000000

Saved metrics to: /content/drive/MyDrive/NLP assignmet/outputs/metrics_summary.csv

Saved report to: /content/drive/MyDrive/NLP assignmet/outputs/classification_report_RF_plus_RandomUnderSampler.csv

Saved confusion matrix to: /content/drive/MyDrive/NLP assignmet/figures/confusion_matrix_RF_plus_RandomUnderSampler.png

RF + SMOTE

After resampling: Counter({'positive': 1399, 'negative': 1399})

Best params: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}

CV best score (f1_weighted): 0.9019

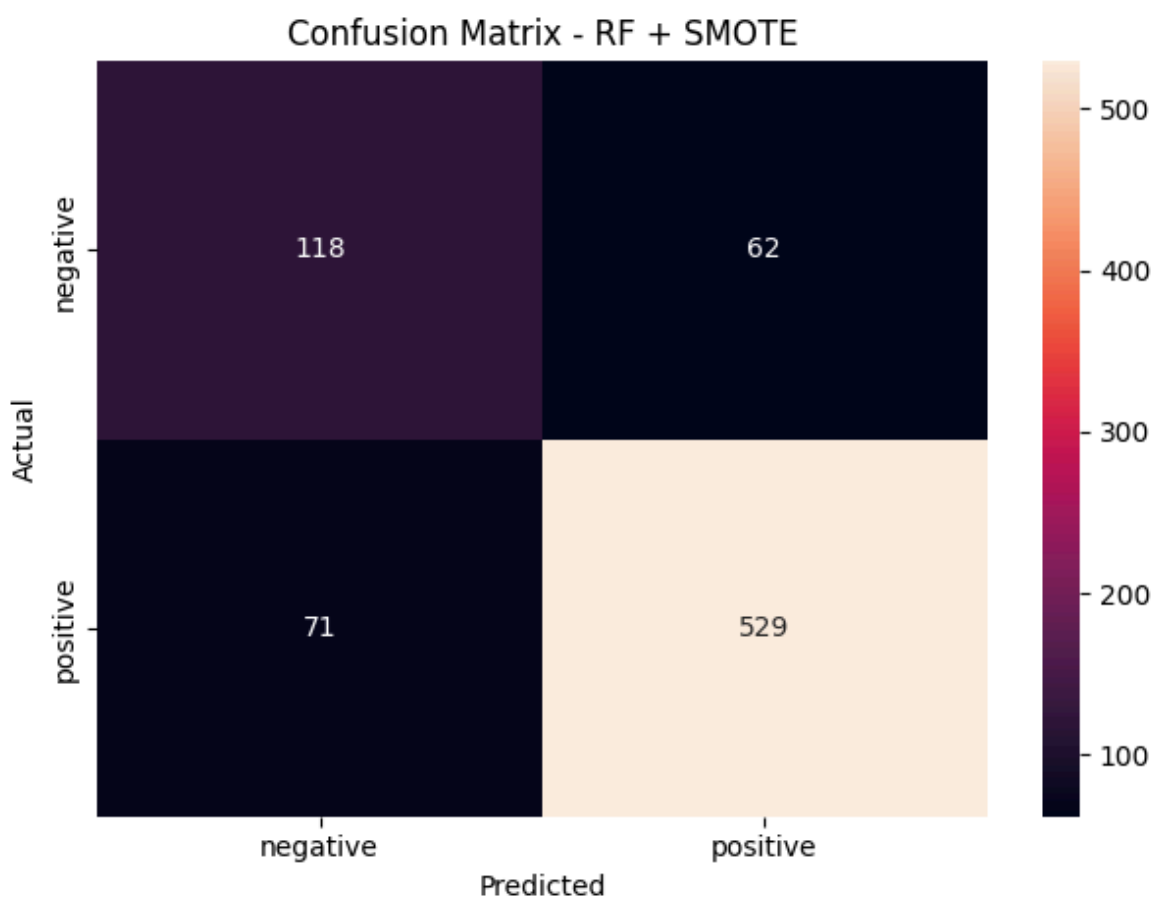
=====

RF + SMOTE

	Accuracy	Precision_weighted	Recall_weighted	F1_weighted
Model				
RF + SMOTE	0.829487	0.832611	0.829487	0.830922

Confusion Matrix:

	negative	positive
negative	118	62
positive	71	529



Classification Report:

	precision	recall	f1-score	support
negative	0.624339	0.655556	0.639566	180.000000
positive	0.895093	0.881667	0.888329	600.000000
accuracy	0.829487	0.829487	0.829487	0.829487
macro avg	0.759716	0.768611	0.763948	780.000000
weighted avg	0.832611	0.829487	0.830922	780.000000

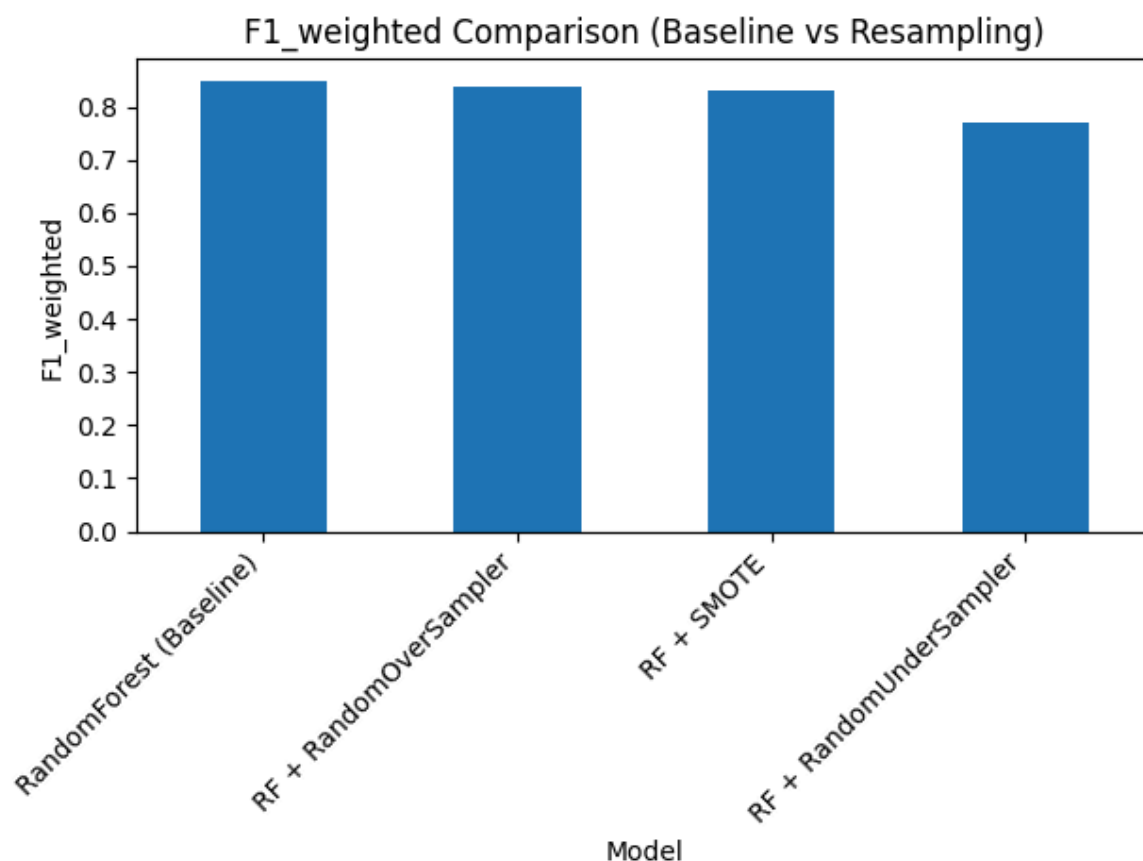
Saved metrics to: /content/drive/MyDrive/NLP assignmet/outputs/metrics_summary.csv

Saved report to: /content/drive/MyDrive/NLP assignmet/outputs/classification_report_RF_plus_SMOTE.csv

Saved confusion matrix to: /content/drive/MyDrive/NLP assignmet/figures/confusion_matrix_RF_plus_SMOTE.png

✅ Final comparison table:

	Accuracy	Precision_weighted	Recall_weighted	F1_weighted
Model				
RandomForest (Baseline)	0.857692	0.850994	0.857692	0.847533
RF + RandomOverSampler	0.835897	0.843783	0.835897	0.839091
RF + SMOTE	0.829487	0.832611	0.829487	0.830922
RF + RandomUnderSampler	0.752564	0.837891	0.752564	0.770999



Saved plot: /content/drive/MyDrive/NLP assignmet/figures/f1_comparison_baseline_vs_resampling.png

Conclusion:

Conclusion

This notebook satisfies the assignment requirements by completing the full NLP classification workflow.

(1) Apply data preprocessing: the raw text was cleaned and normalized (lowercasing, removing URLs/HTML/special characters, tokenization, stopword removal, and normalization) to generate a `clean_text` feature.

(2) Split the dataset: the dataset was split into training and testing sets using a stratified split to preserve the original class distribution.

(3) Apply feature representation methods: TF-IDF was used to convert the preprocessed text into numerical feature vectors.

(4) Train and evaluate models: a Random Forest classifier was trained and optimized using 5-fold cross-validation with hyperparameter tuning, and evaluation was performed using accuracy, precision, recall, weighted F1-score, classification reports, and confusion matrices. Additionally, resampling methods (RandomOverSampler, RandomUnderSampler, and SMOTE) were applied to assess the impact of class imbalance handling on model performance.

The baseline Random Forest achieved **Accuracy = 0.858** and **weighted F1 = 0.848**, but minority-class performance was weaker (**negative recall = 0.533**). After imbalance handling, RandomOverSampler improved minority detection (**negative recall = 0.706**) with a moderate trade-off in overall accuracy (**0.836**), RandomUnderSampler achieved the highest minority recall (**0.861**) but reduced overall accuracy (**0.753**), and SMOTE produced intermediate results (**Accuracy = 0.829, negative recall = 0.656**). All results (metrics tables, reports, and confusion matrix plots) were saved to the project folders in Google Drive, and the trained model and TF-IDF vectorizer were saved under the `models/` directory.

In []: