

```
In [12]: # Import Libraries

import pandas as pd
import numpy as np
import re

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

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

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

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

from collections import Counter
```

```
In [14]: # Download NLTK Resources

nltk.download('punkt', quiet=True)
nltk.download('punkt_tab', quiet=True)
nltk.download('stopwords', quiet=True)
```

```
Out[14]: True
```

```
In [16]: # Function to Evaluate Model Performance

def evaluate_model(model_name, y_true, y_pred):
    """
    Evaluates a classification model and returns common performance metrics.
    """
    metrics = {
        "Model Name": model_name,
        "Accuracy": accuracy_score(y_true, y_pred),
        "Precision": precision_score(y_true, y_pred, average="weighted", zero_division=0),
        "Recall": recall_score(y_true, y_pred, average="weighted", zero_division=0),
        "F1 Score": f1_score(y_true, y_pred, average="weighted", zero_division=0),
        "Confusion Matrix": confusion_matrix(y_true, y_pred),
        "Classification Report": classification_report(y_true, y_pred, zero_division=0)
    }
    return metrics
```

```
In [18]: def train_random_forest_classifier_with_grid_search(
    X_train_vec, y_train, X_test_vec, y_test, evaluate_model_func
):
    rf_classifier = RandomForestClassifier(random_state=42)

    param_grid = {
        "n_estimators": [100, 200, 300],
        "max_depth": [None, 10, 20, 30]
    }

    grid_search = GridSearchCV(
        estimator=rf_classifier,
        param_grid=param_grid,
        cv=5,
        scoring="accuracy",
        n_jobs=-1
    )

    grid_search.fit(X_train_vec, y_train)

    best_rf = grid_search.best_estimator_
    y_pred = best_rf.predict(X_test_vec)

    evaluation_results = evaluate_model_func(
        "RandomForestClassifier", y_test, y_pred
    )

    print("==== Random Forest Evaluation ===")
    for key, value in evaluation_results.items():
        if key == "Classification Report":
            print("\nClassification Report:\n", value)
        elif key == "Confusion Matrix":
            print("\nConfusion Matrix:\n", value)
        elif isinstance(value, float):
            print(f"{key}: {value:.4f}")
        else:
            print(f"{key}: {value}")

    print("\nBest Hyperparameters:")
    print(grid_search.best_params_)

    return evaluation_results
```

```
In [10]: # Read Dataset

df = pd.read_csv("unbalanceddataset.csv")

print(df.head())
print(f"\nClass Distribution:\n", df["sentiment"].value_counts())

# 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

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

```
In [20]: # Preprocessing

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

def preprocess_text(text):
    """
    Cleans and preprocesses English text:
    - Lowercasing
    - Removing URLs and HTML tags
    - Removing special characters and numbers
    - Tokenization
    - Stopword removal
    - Stemming
    """
    text = str(text).lower()
    text = re.sub("https?://www\S+", "", text)
    text = re.sub("[<.+>]", "", text)
    text = re.sub("[a-zA-Z5]", "", text)

    tokens = word_tokenize(text)
    tokens = [word for word in tokens if word not in stop_words]
    tokens = [stemmer.stem(word) for word in tokens]

    return " ".join(tokens)

df["clean_text"] = df["text"].apply(preprocess_text)
df = df.dropna(subset=["clean_text"])

df[["text", "sentiment", "clean_text"]].head()
```

```
Out[20]:
```

	text	sentiment	clean_text
0	Java Concurrency in Practice is probably the b...	positive	java concurr practic probabl best java book k...
1	haha aww hun i bet you are more creative tha...	positive	haha aww hun bet creativ
2	_pickle lol, thank you very much Hope you're h...	positive	pickl lol thank much hope your great day
3	Out for an evening on the town with jeremy. Sa...	negative	even town jeremi sad carri cant come
4	- just took over the #1 Most Endorsed spot on...	positive	took endors spot twindexcom thank endors

```
In [14]: # Split Dataset + Feature Representation (TF-IDF)

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

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

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

print(f"Train distribution:\n", y_train.value_counts())
print(f"\nTest distribution:\n", y_test.value_counts())

Train distribution:
sentiment
positive    1400
negative    420
Name: count, dtype: int64

Test distribution:
sentiment
positive    600
negative    180
Name: count, dtype: int64
```

```
In [22]: # Baseline Random Forest Classifier

print("\n---- Baseline Random Forest ----")
rf_base_results = train_random_forest_classifier_with_grid_search(
    X_train_vec, y_train, X_test_vec, y_test, evaluate_model
)

---- Baseline Random Forest ----

=== Random Forest Evaluation ===
Model Name: RandomForestClassifier
Accuracy: 0.8487
Precision: 0.8463
Recall: 0.8487
F1 Score: 0.8300

Confusion Matrix:
[[ 78 102]
 [ 16 584]]

Classification Report:
              precision    recall  f1-score   support

negative      0.83      0.43      0.57      180
positive      0.85      0.97      0.91      600

accuracy              0.84      0.70      0.85      780
macro avg              0.84      0.70      0.74      780
weighted avg              0.85      0.85      0.83      780

Best Hyperparameters:
{'max_depth': None, 'n_estimators': 100}
```

```
In [30]: # RandomOverSampler + Random Forest

ros = RandomOverSampler(random_state=42)
X_ros, y_ros = ros.fit_resample(X_train_vec, y_train)

print("After RandomOverSampling:", Counter(y_ros))

print("\n---- Random Forest with RandomOverSampler ----")
rf_ros_results = train_random_forest_classifier_with_grid_search(
    X_ros, y_ros, X_test_vec, y_test, evaluate_model
)

After RandomOverSampling: Counter({'positive': 1400, 'negative': 1400})

---- Random Forest with RandomOverSampler ----

=== Random Forest Evaluation ===
Model Name: RandomForestClassifier
Accuracy: 0.8551
Precision: 0.8478
Recall: 0.8551
F1 Score: 0.8482

Confusion Matrix:
[[103  77]
 [ 36 564]]

Classification Report:
              precision    recall  f1-score   support

negative      0.74      0.57      0.65      180
positive      0.88      0.94      0.91      600

accuracy              0.81      0.76      0.86      780
macro avg              0.81      0.76      0.78      780
weighted avg              0.85      0.86      0.85      780

Best Hyperparameters:
{'max_depth': None, 'n_estimators': 300}
```

```
In [32]: # RandomUnderSampler + Random Forest

rus = RandomUnderSampler(random_state=42)
X_rus, y_rus = rus.fit_resample(X_train_vec, y_train)

print("After RandomUnderSampling:", Counter(y_rus))

print("\n---- Random Forest with RandomUnderSampler ----")
rf_rus_results = train_random_forest_classifier_with_grid_search(
    X_rus, y_rus, X_test_vec, y_test, evaluate_model
)

After RandomUnderSampling: Counter({'negative': 420, 'positive': 420})

---- Random Forest with RandomUnderSampler ----

=== Random Forest Evaluation ===
Model Name: RandomForestClassifier
Accuracy: 0.7154
Precision: 0.8397
Recall: 0.7154
F1 Score: 0.7378

Confusion Matrix:
[[163  17]
 [205 395]]

Classification Report:
              precision    recall  f1-score   support

negative      0.44      0.91      0.59      180
positive      0.96      0.66      0.78      600

accuracy              0.70      0.78      0.72      780
macro avg              0.70      0.78      0.69      780
weighted avg              0.84      0.72      0.74      780

Best Hyperparameters:
{'max_depth': 30, 'n_estimators': 300}
```

```
In [34]: # Summary Table for Comparison

summary_df = pd.DataFrame([
    {"Method": "Baseline", "Accuracy": rf_base_results["Accuracy"], "F1": rf_base_results["F1 Score"]},
    {"Method": "RandomOverSampler", "Accuracy": rf_ros_results["Accuracy"], "F1": rf_ros_results["F1 Score"]},
    {"Method": "RandomUnderSampler", "Accuracy": rf_rus_results["Accuracy"], "F1": rf_rus_results["F1 Score"]},
])

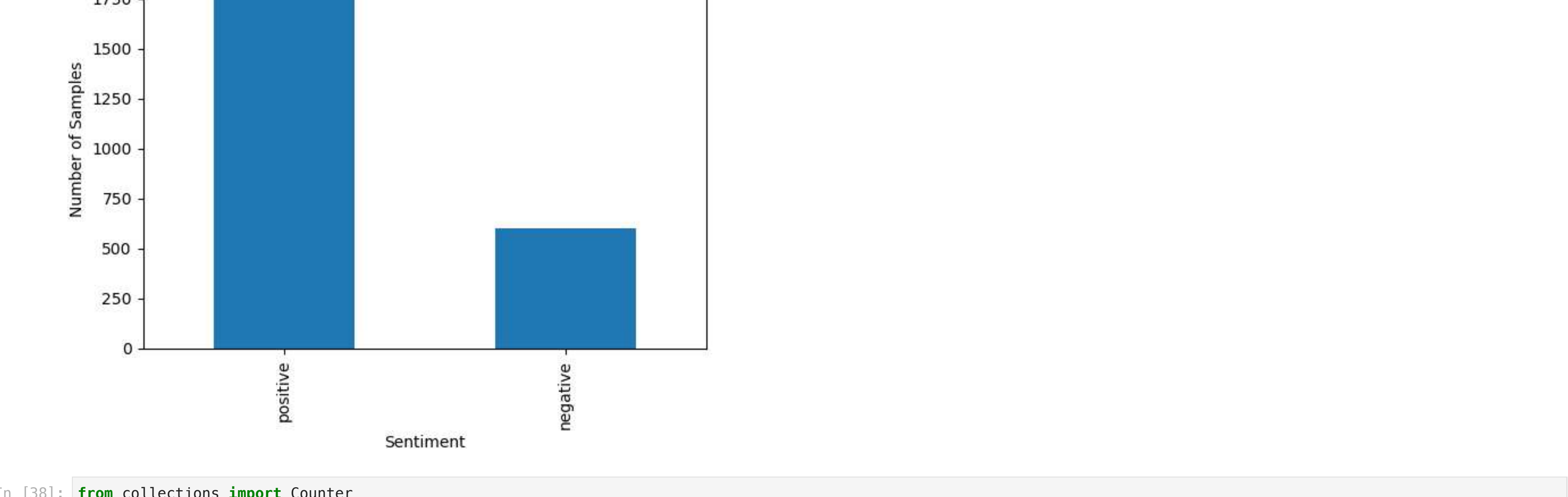
summary_df
```

```
Out[34]:
```

	Method	Accuracy	F1
0	Baseline	0.848718	0.830035
1	RandomOverSampler	0.855128	0.848211
2	RandomUnderSampler	0.715385	0.737769

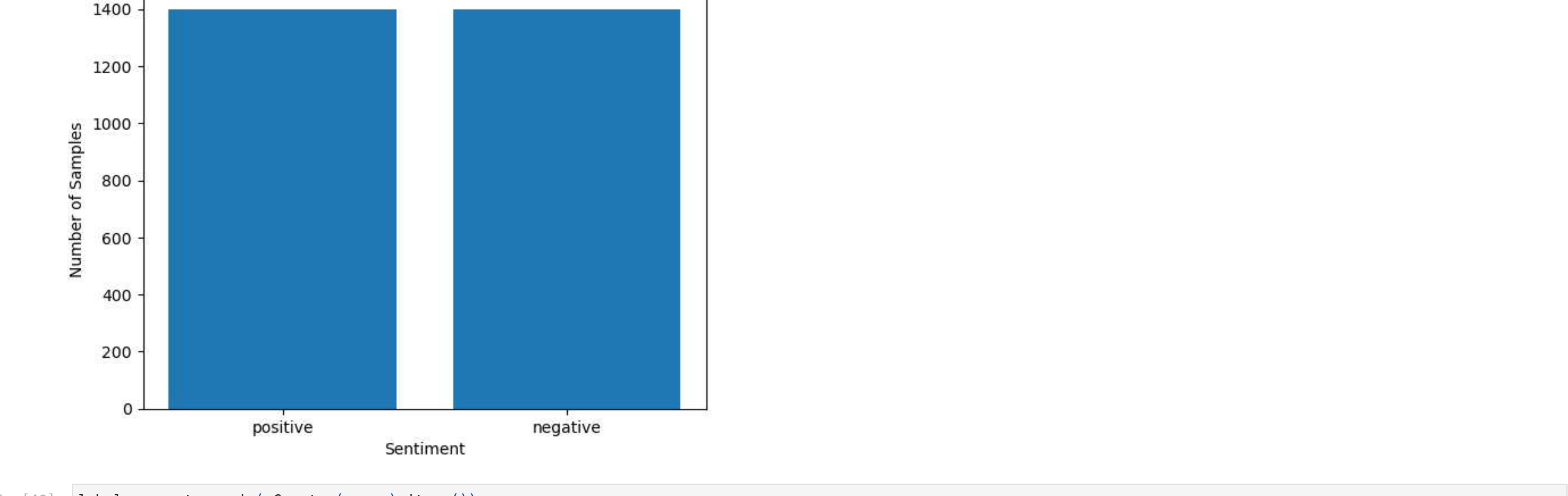
```
In [36]: import matplotlib.pyplot as plt

df["sentiment"].value_counts().plot(kind='bar')
plt.title("Class Distribution Before Resampling")
plt.xlabel("Sentiment")
plt.ylabel("Number of Samples")
plt.show()
```

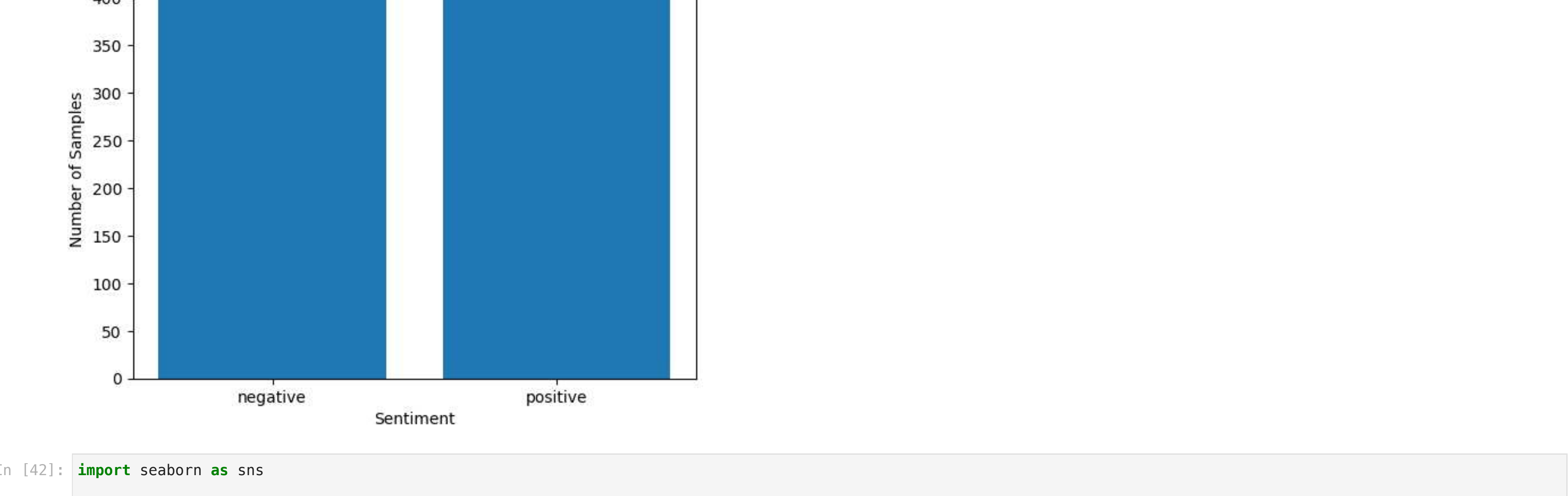


```
In [38]: from collections import Counter

labels, counts = zip(*Counter(y_ros).items())
plt.bar(labels, counts)
plt.title("Class Distribution After RandomOverSampling")
plt.xlabel("Sentiment")
plt.ylabel("Number of Samples")
plt.show()
```

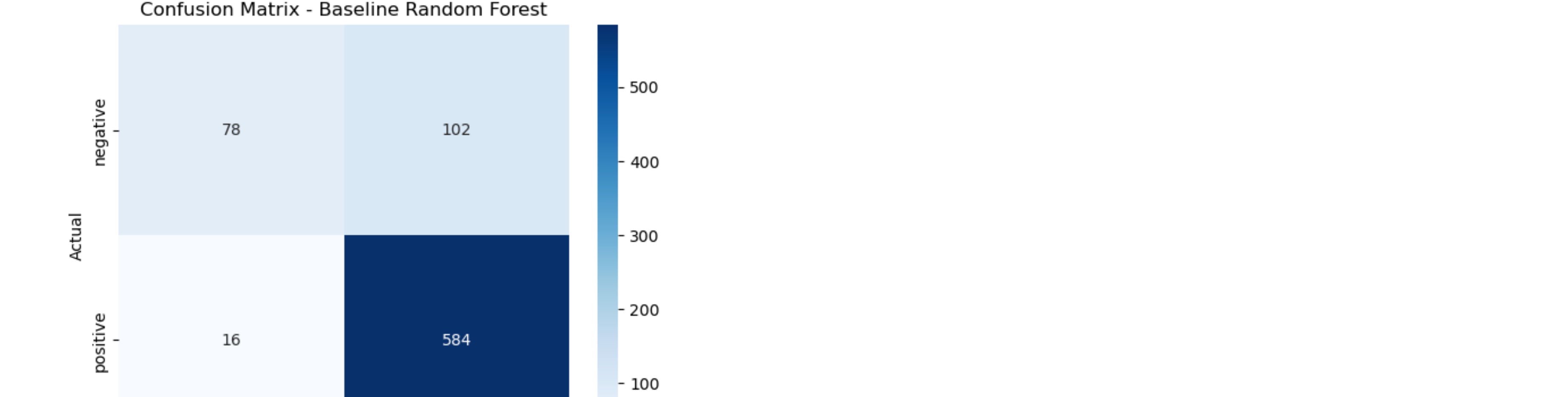


```
In [40]: labels, counts = zip(*Counter(y_rus).items())
plt.bar(labels, counts)
plt.title("Class Distribution After RandomUnderSampling")
plt.xlabel("Sentiment")
plt.ylabel("Number of Samples")
plt.show()
```

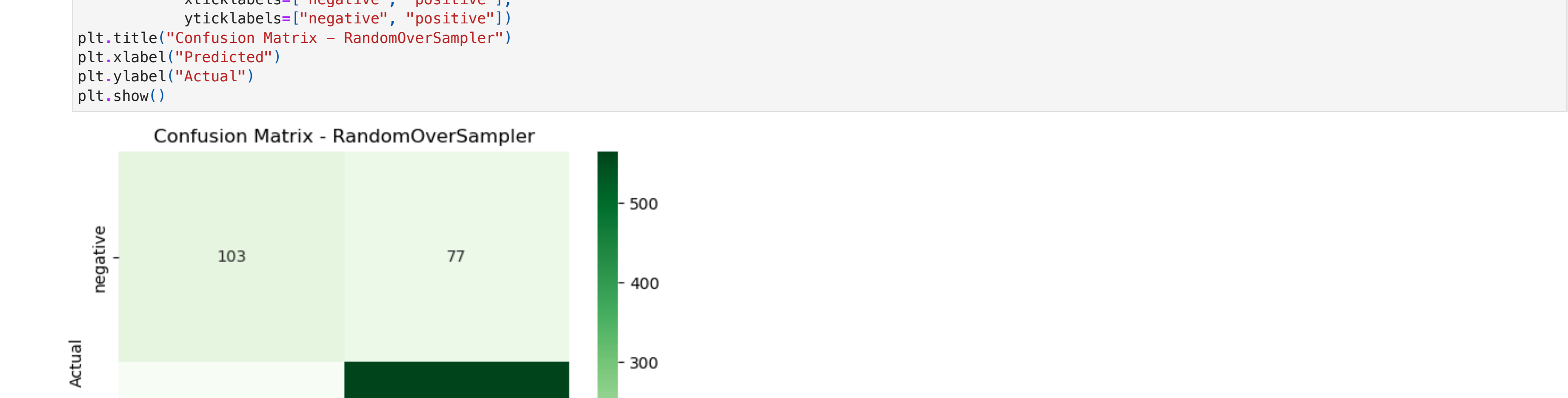


```
In [42]: import seaborn as sns

cm = rf_base_results["Confusion Matrix"]
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=["negative", "positive"],
            yticklabels=["negative", "positive"])
plt.title("Confusion Matrix - Baseline Random Forest")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



```
In [44]: cm = rf_ros_results["Confusion Matrix"]
sns.heatmap(cm, annot=True, fmt="d", cmap="Greens",
            xticklabels=["negative", "positive"],
            yticklabels=["negative", "positive"])
plt.title("Confusion Matrix - RandomOverSampler")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



```
In [ ]:
```