

NLP Classification Project

```
import pandas as pd

df = pd.read_csv('/clean_dataset (1).csv')

print("Dataset Shape:", df.shape)
df.head()
```

Dataset Shape: (19557, 2)

	sentiment	clean_text	
0	negative	sooo sad miss san diego	
1	negative	boss bulli	
2	negative	interview leav alon	
3	negative	son couldnt put releas already bought	
4	positive	feed babi fun smile coo	

Next steps:

[Generate code with df](#)
[New interactive sheet](#)

```
# Check missing values
print("Missing values:")
print(df.isnull().sum())

# Check duplicate rows
print("\nDuplicate rows:", df.duplicated().sum())

# Check sentiment distribution
print("\nSentiment distribution:")
print(df["sentiment"].value_counts())
```

```
Missing values:
sentiment      0
clean_text     22
dtype: int64
```

```
Duplicate rows: 404
```

```
Sentiment distribution:
sentiment
positive      8582
negative      7781
neutral       3194
Name: count, dtype: int64
```

```
# Data preprocessing
```

```
# Remove rows with missing text
```

```
df = df.dropna(subset=["clean_text"])

# Remove duplicate rows
df = df.drop_duplicates()

# Remove empty text rows
df = df[df["clean_text"].str.strip() != ""]

# Reset index
df = df.reset_index(drop=True)

# Check the dataset after preprocessing
print("Dataset shape after preprocessing:", df.shape)

print("\nMissing values:")
print(df.isnull().sum())

print("\nDuplicate rows:", df.duplicated().sum())

print("\nSentiment distribution:")
print(df["sentiment"].value_counts())
```

Dataset shape after preprocessing: (19150, 2)

Missing values:
sentiment 0
clean_text 0
dtype: int64

Duplicate rows: 0

Sentiment distribution:
sentiment
positive 8331
negative 7666
neutral 3153
Name: count, dtype: int64

```
from sklearn.model_selection import train_test_split

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

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.20,
    random_state=42,
    stratify=y
)

print("Training samples:", len(X_train))
print("Testing samples:", len(X_test))

print("\nTraining label distribution:")
```

```
print(y_train.value_counts())

print("\nTesting label distribution:")
print(y_test.value_counts())
```

```
Training samples: 15320
Testing samples: 3830

Training label distribution:
sentiment
positive    6665
negative    6133
neutral     2522
Name: count, dtype: int64

Testing label distribution:
sentiment
positive    1666
negative    1533
neutral      631
Name: count, dtype: int64
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Create TF-IDF vectorizer
tfidf = TfidfVectorizer(
    max_features=5000,
    ngram_range=(1, 2)
)

# Fit only on training data
X_train_tfidf = tfidf.fit_transform(X_train)

# Transform test data
X_test_tfidf = tfidf.transform(X_test)

print("Training TF-IDF shape:", X_train_tfidf.shape)
print("Testing TF-IDF shape:", X_test_tfidf.shape)
```

```
Training TF-IDF shape: (15320, 5000)
Testing TF-IDF shape: (3830, 5000)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

# -----
# 1. Logistic Regression
# -----
lr_model = LogisticRegression(
    max_iter=1000,
    random_state=42
)
```

```

lr_model.fit(X_train_tfidf, y_train)

lr_pred = lr_model.predict(X_test_tfidf)

print("==== Logistic Regression =====")
print("Accuracy:", accuracy_score(y_test, lr_pred))
print(classification_report(y_test, lr_pred))

# -----
# 2. Linear SVM
# -----
svm_model = LinearSVC(
    random_state=42
)

svm_model.fit(X_train_tfidf, y_train)

svm_pred = svm_model.predict(X_test_tfidf)

print("==== Linear SVM =====")
print("Accuracy:", accuracy_score(y_test, svm_pred))
print(classification_report(y_test, svm_pred))

# -----
# 3. Naive Bayes
# -----
nb_model = MultinomialNB()

nb_model.fit(X_train_tfidf, y_train)

nb_pred = nb_model.predict(X_test_tfidf)

print("==== Naive Bayes =====")
print("Accuracy:", accuracy_score(y_test, nb_pred))
print(classification_report(y_test, nb_pred))

```

```

==== Logistic Regression =====
Accuracy: 0.7407310704960836

```

	precision	recall	f1-score	support
negative	0.73	0.84	0.78	1533
neutral	0.47	0.17	0.25	631
positive	0.78	0.86	0.82	1666
accuracy			0.74	3830
macro avg	0.66	0.62	0.62	3830
weighted avg	0.71	0.74	0.71	3830

```

==== Linear SVM =====
Accuracy: 0.7297650130548303

```

	precision	recall	f1-score	support
negative	0.75	0.79	0.77	1533
neutral	0.43	0.28	0.34	631

positive	0.78	0.85	0.81	1666
accuracy			0.73	3830
macro avg	0.65	0.64	0.64	3830
weighted avg	0.71	0.73	0.72	3830

===== Naive Bayes =====

Accuracy: 0.7182767624020888

	precision	recall	f1-score	support
negative	0.71	0.85	0.78	1533
neutral	0.25	0.00	0.01	631
positive	0.73	0.86	0.79	1666
accuracy			0.72	3830
macro avg	0.56	0.57	0.52	3830
weighted avg	0.64	0.72	0.66	3830

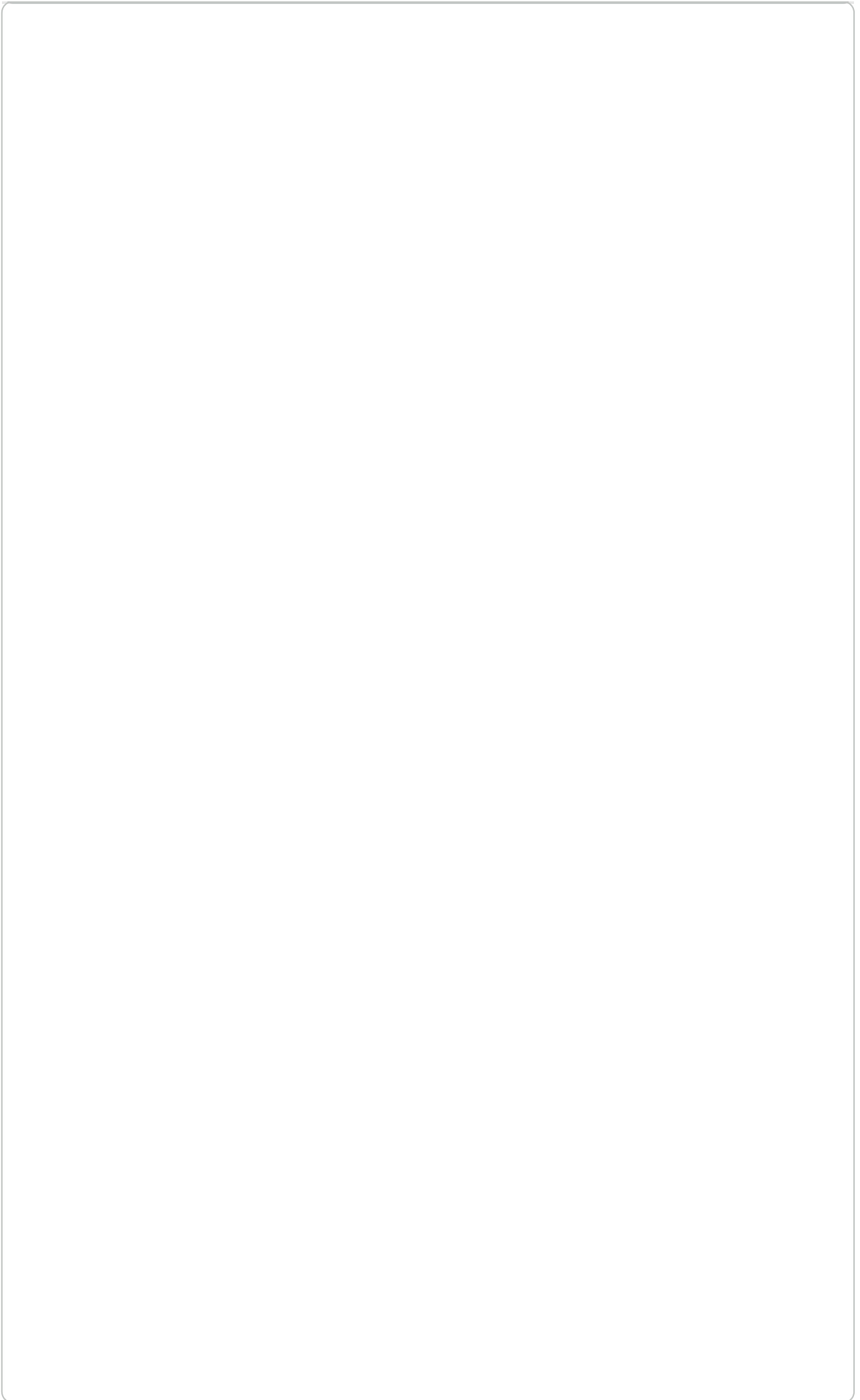
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

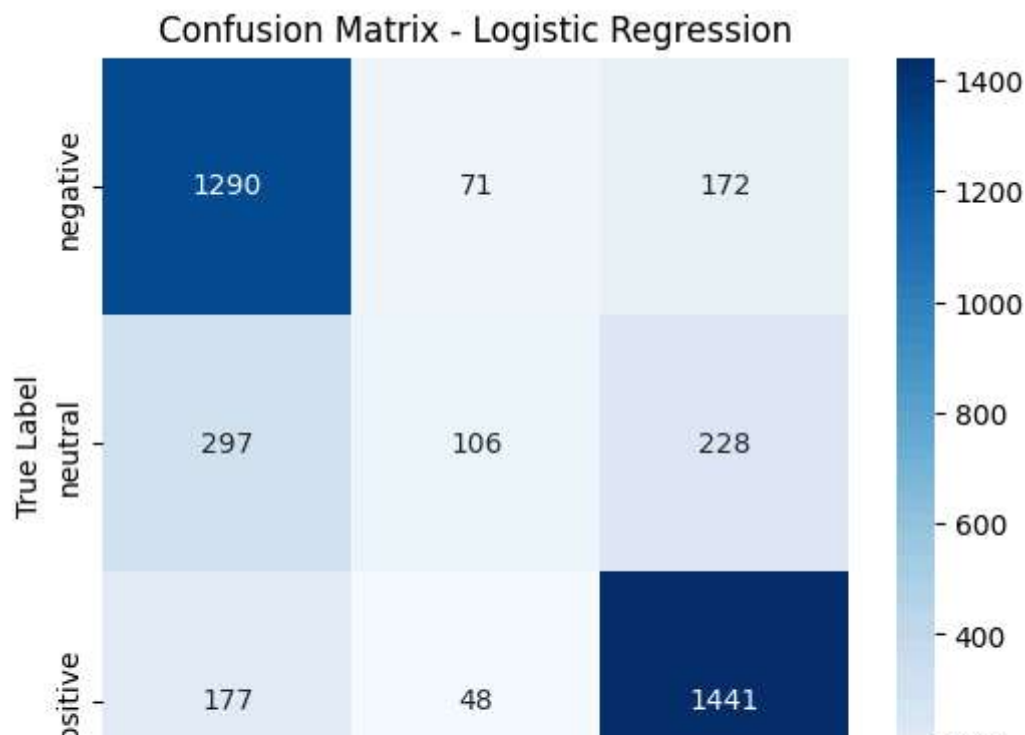
models_predictions = {
    "Logistic Regression": lr_pred,
    "Linear SVM": svm_pred,
    "Naive Bayes": nb_pred
}

for model_name, predictions in models_predictions.items():
    cm = confusion_matrix(y_test, predictions, labels=["negative", "neutral", "positive"])

    plt.figure(figsize=(6, 5))
    sns.heatmap(
        cm,
        annot=True,
        fmt="d",
        cmap="Blues",
        xticklabels=["negative", "neutral", "positive"],
        yticklabels=["negative", "neutral", "positive"]
    )

    plt.title(f"Confusion Matrix - {model_name}")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()
```





```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import pandas as pd
```

```
results = pd.DataFrame({
    "Model": [
        "Logistic Regression",
        "Linear SVM",
        "Naive Bayes"
    ],
    "Accuracy": [
        accuracy_score(y_test, lr_pred),
        accuracy_score(y_test, svm_pred),
        accuracy_score(y_test, nb_pred)
    ],
    "Macro Precision": [
        precision_score(y_test, lr_pred, average="macro"),
        precision_score(y_test, svm_pred, average="macro"),
        precision_score(y_test, nb_pred, average="macro")
    ],
    "Macro Recall": [
        recall_score(y_test, lr_pred, average="macro"),
        recall_score(y_test, svm_pred, average="macro"),
        recall_score(y_test, nb_pred, average="macro")
    ],
    "Macro F1": [
        f1_score(y_test, lr_pred, average="macro"),
        f1_score(y_test, svm_pred, average="macro"),
        f1_score(y_test, nb_pred, average="macro")
    ]
})
```

```
results = results.sort_values(by="Accuracy", ascending=False)
```

```
results
```

	Model	Accuracy	Macro Precision	Macro Recall	Macro F1
0	Logistic Regression	0.740731	0.661710	0.624807	0.617326
1	Linear SVM	0.729765	0.653353	0.637573	0.639192
2	Naive Bayes	0.718277	0.562658	0.573782	0.523965

New interactive sheet

negative	0.74	0.80	0.77	1533
----------	------	------	------	------

neutral	0.43	0.26	0.33	631
positive	0.79	0.84	0.81	1666
accuracy			0.73	3830
macro avg	0.65	0.64	0.64	3830
weighted avg	0.71	0.73	0.72	3830

```
# Add Bag of Words result to the comparison table

bow_result = pd.DataFrame({
    "Model": ["Logistic Regression - BoW"],
    "Accuracy": [accuracy_score(y_test, bow_lr_pred)],
    "Macro Precision": [precision_score(y_test, bow_lr_pred, average="macro")],
    "Macro Recall": [recall_score(y_test, bow_lr_pred, average="macro")],
    "Macro F1": [f1_score(y_test, bow_lr_pred, average="macro")]
})

final_results = pd.concat([results, bow_result], ignore_index=True)

final_results = final_results.sort_values(
    by="Accuracy",
    ascending=False
).reset_index(drop=True)

final_results
```

	Model	Accuracy	Macro Precision	Macro Recall	Macro F1
0	Logistic Regression	0.740731	0.661710	0.624807	0.617326
1	Logistic Regression - BoW	0.731070	0.651998	0.636036	0.636839
2	Linear SVM	0.729765	0.653353	0.637573	0.639192
3	Naive Bayes	0.718277	0.562658	0.573782	0.523965

Next steps: [Generate code with final_results](#)

[New interactive sheet](#)