# 📓 Assignment 1: Sentiment Classification of Twitter Tweets (NLP Pipeline)

🧠 **Difficulty**: Advanced | 🕒 **Time**: 4–5 hours
📦 **Dataset**: Twitter US Airline Sentiment
📚 **Tools**: Python, NLTK/spaCy, Scikit-learn, Pandas, NumPy, Seaborn, Matplotlib, Streamlit or MLflow

## 🔍 Problem Statement

You are working for a customer support analytics team at an airline company. Your task is to **build a sentiment classifier** that can automatically categorize tweets about airline services into **positive, negative, or neutral**. The solution should be modular and support continuous improvements and deployment.

## ☑ Assignment Objectives

1. Build a complete ML pipeline to classify tweets using TF-IDF + traditional ML models (Logistic Regression / Naive Bayes).
2. Use OOP principles to structure code into reusable classes/functions.
3. Deploy the model with a frontend using Streamlit or register and track it using MLflow.

## 🧱 Task Breakdown

### *Task 1: Data Cleaning & Exploration*

- Load and explore the dataset
- Handle missing values
- Visualize class imbalance

### *Task 2: Preprocessing with OOP*

- Create a class `TweetPreprocessor`:
  - `clean_text()`: remove mentions, links, emojis
  - `tokenize_and_lemmatize()`: using spaCy or NLTK
  - `remove_stopwords()` method

### *Task 3: Modeling Pipeline*

- Create a class `SentimentModel`:
  - `vectorize()` using TF-IDF
  - `train_model()` using Logistic Regression
  - `evaluate()` using F1, accuracy, confusion matrix

### *Task 4: Deployment*

✅ **Option A – Streamlit**

- User enters tweet → predicted sentiment is shown

✅ **Option B – MLflow**

- Track model:
  - Preprocessing steps
  - Model accuracy
  - Parameters
- Register best model

## 📅 Deliverables

- Python notebook or scripts using modular OOP
- TF-IDF vectorizer saved
- Deployment (Streamlit or MLflow)
- Screenshot of model evaluation or UI
- README with setup and usage

# 📓 Assignment 2: News Topic Classification Using BERT (Transformer-based NLP)

🧠 **Difficulty**: Expert | 🕒 **Time**: 5–6 hours
📦 **Dataset**: AG News Classification Dataset on Kaggle
📚 **Tools**: HuggingFace Transformers, PyTorch or TensorFlow, Pandas, Seaborn, Matplotlib, Streamlit or MLflow

## 🔍 Problem Statement

A media analytics company wants to categorize news articles automatically into one of 4 categories: **World, Sports, Business, and Sci/Tech**. You are asked to build a transformer-based pipeline using **BERT** to improve classification accuracy over traditional methods.

## ✅ Assignment Objectives

1. Use HuggingFace's BERT model for fine-tuning on multi-class classification.
2. Implement reusable classes/functions for tokenization, modeling, and prediction.
3. Integrate experiment tracking or UI deployment using MLflow or Streamlit.

## 🧱 Task Breakdown

### *Task 1: Data Loading and Cleaning*

- Load train/test CSVs
- Basic text cleanup (remove special characters, optional)

### *Task 2: Tokenization & Encoding*

- Create class `NewsTokenizer`:
    - Load BERT tokenizer

    - Tokenize and pad sequences
    - Encode target labels

### *Task 3: Model Training*

- Create class `NewsClassifier`:
    - Load `bert-base-uncased`
    - Freeze/unfreeze layers
    - Train with learning rate scheduling and validation loop

### *Task 4: Evaluation & Metrics*

- Use F1 score, confusion matrix, and classification report

### *Task 5: Deployment*

#### ✅ Option A – Streamlit

- Upload a text → category is predicted
- Dropdown to select pre-trained or fine-tuned model

#### ✅ Option B – MLflow

- Log:
    - Training time
    - Accuracy, F1
    - Model name, tokenizer, parameters
- Save and register model artifact

## 🗃️ Deliverables

- Scripts: `tokenizer.py`, `bert_model.py`, `utils.py`
- Saved model and tokenizer
- Streamlit or MLflow deployment
- Instructions in README

## ✅ Bonus: Instructions Format for Submission

Each assignment should follow this directory structure:

```
📂 assignment_name/
│
├── 📁 data/            # Store dataset here
├── 📁 models/          # Saved models or tokenizer
├── 📁 app/             # Streamlit or Flask app (optional)
├── 📄 main.py          # Main runner file
├── 📄 README.md        # Setup, usage, deployment instructions
└── 📄 requirements.txt    # List of dependencies
```