Q1: TensorFlow vs. PyTorch

TensorFlow:

Static computation graph (define-then-run), optimized for deployment.
Strong production tools (TF Serving, Lite).
Ideal for large-scale systems and mobile deployment.

PyTorch:

Dynamic computation graph (define-by-run), easier debugging.
Pythonic syntax, preferred for research/prototyping.

When to choose:

TensorFlow: Production pipelines, edge devices.
PyTorch: Rapid experimentation, academia.

Q2: Jupyter Notebook Use Cases

Interactive Prototyping:

Visualize data/model outputs (e.g., matplotlib plots) during exploratory analysis.

Educational Demos:

Combine code, equations, and explanations (Markdown) to teach ML concepts.

Q3: spaCy vs. Basic String Operations

spaCy:

Pre-trained models for POS tagging, dependency parsing, and NER.
Handles linguistic nuances (e.g., "Apple" as company vs. fruit).

Basic Strings:

Limited to regex/string matching; fails at context (e.g., irony in "This 'gift' ruined my day").

Comparative Analysis: Scikit-learn vs. TensorFlow

Scikit-learn:

Target Applications:	Classical ML (SVM, decision trees)
Ease of Use:	Simple API, less code for traditional ML
Community Support:	Extensive docs/tutorials for ML basics

TensorFlow:

Target Applications:	Deep Learning (CNNs, RNNs)
Ease of Use:	Steeper learning curve; flexible for DL
Community Support:	Massive ecosystem; industry resources