Q1: Differences Between TensorFlow and PyTorch

-

Eager vs. Graph Execution: PyTorch uses *eager execution* (operations run immediately), making it intuitive and Pythonic. TensorFlow originally used static computation graphs but now supports eager execution via `tf.function`.

-
-

Syntax and Debugging: PyTorch is often easier to debug due to its dynamic nature. TensorFlow, however, has more industrial deployment tools (e.g., TensorFlow Lite, TensorFlow Serving).

-
-

When to choose:

-
    -

PyTorch: Preferred for rapid research and prototyping.

    -
    -

TensorFlow: Often used in production environments requiring high scalability.

    -

Q2: Use Cases for Jupyter Notebooks in AI

-

Interactive Prototyping: Allows researchers to quickly test and visualize model performance, tweak hyperparameters, and visualize datasets or loss functions inline.

-
-

Documentation and Reporting: Enables mixing code, charts, and markdown to explain AI workflows to stakeholders or team members.

-

Q3: spaCy vs. Basic Python String Operations

-

Pre-trained Models: spaCy uses statistical models for tokenization, part-of-speech tagging, named entity recognition (NER), and dependency parsing—tasks not achievable with basic `str` methods.

-
-

Efficiency and Accuracy: spaCy handles language-specific rules and ambiguity more accurately and efficiently than regex/string manipulation, especially in real-world texts.

-

---

2. Comparative Analysis: Scikit-learn vs. TensorFlow

Criteria	Scikit-learn	TensorFlow
Target Applications	Classical ML (SVM, Random Forest, etc.)	Deep Learning (CNNs, RNNs, Transformers)
Ease for Beginners	Very beginner-friendly; simple API	Steeper learning curve; more flexible
Community Support	Large, mature community; widely used	Extensive community + Google support