**1. Short Answer Questions**

**Q1: Explain the primary differences between TensorFlow and PyTorch. When would you choose one over the other?**

The primary difference is their development philosophy and ecosystem. PyTorch is more "Pythonic" with a define-by-run (dynamic) approach, making it feel intuitive and excellent for rapid prototyping and research. TensorFlow, especially with its Keras API, emphasizes a more structured define-and-run paradigm and offers a more extensive, production-ready ecosystem (TFX, TF Serving, TF Lite).

* **Choose PyTorch for:** Research, rapid prototyping, and projects where a flexible, imperative coding style is preferred.
* **Choose TensorFlow for:** Large-scale production deployments, mobile and edge computing (via TF Lite), and when leveraging its comprehensive end-to-end ecosystem is a priority.

**Q2: Describe two use cases for Jupyter Notebooks in AI development.**

1. **Exploratory Data Analysis (EDA):** Notebooks are ideal for interactively loading, cleaning, and visualizing datasets. The cell-based execution allows developers to iteratively run code snippets and immediately see outputs like dataframes, plots, and statistical summaries, which is crucial for understanding data before modeling.
2. **Model Prototyping and Iteration:** Notebooks provide a fast feedback loop for building and training models. A developer can define a model architecture in one cell, train it in the next, and visualize results (e.g., loss curves) in another, all within the same document. This makes it easy to tweak hyperparameters and experiment with different architectures rapidly.

**Q3: How does spaCy enhance NLP tasks compared to basic Python string operations?**

spaCy provides **linguistic structure** where basic string operations see only characters. Instead of simply splitting text on whitespace, spaCy processes text through a pipeline of pre-trained models to create Doc objects containing rich annotations.

Key enhancements include:

* **Accurate Tokenization:** Understands punctuation, prefixes, and suffixes (e.g., treats "U.S.A." as one token).
* **Part-of-Speech (POS) Tagging:** Identifies nouns, verbs, adjectives, etc.
* **Named Entity Recognition (NER):** Extracts entities like "Person," "Organization," and "Date."
* **Lemmatization:** Reduces words to their base form ("running" -> "run").

This transforms raw text into structured, analyzable data, enabling far more sophisticated analysis than str.split() or regular expressions.

**2. Comparative Analysis**

**Scikit-learn vs. TensorFlow**

|  |  |  |
| --- | --- | --- |
| Feature | Scikit-learn | TensorFlow |
| **Target Applications** | **Classical Machine Learning.** Best for tasks on structured/tabular data. Implements algorithms like Linear Regression, Random Forests, SVMs, and K-Means clustering. It is not designed for building deep neural networks from scratch. | **Deep Learning.** The primary tool for building, training, and deploying neural networks (CNNs, RNNs, Transformers). Excels with unstructured data like images, text, and audio. Can handle classical ML, but it's often overkill. |
| **Ease of Use for Beginners** | **Significantly easier.** It features a simple, consistent API (fit(), predict(), transform()) that is easy to learn. The underlying algorithms are more conceptually accessible, making it the standard for learning ML fundamentals. | **Steeper learning curve.** While tf.keras has greatly simplified the API, beginners must still grasp more complex concepts like tensors, layers, activation functions, and optimizers. The level of abstraction is higher. |