QUESTION 1: TensorFlow vs PyTorch Differences

1. Computational Graph Definition:

TensorFlow: Uses static computational graphs.You define the graph first, then execute it.

PyTorch: Uses dynamic computational graphs (define by run)

2. Syntax:

TensorFlow: More declarative style.

PyTorch: More imperative and intuitive, feels like regular Python code.

3. Debugging:

TensorFlow: Debugging static graphs can be challenging.

PyTorch: Easier debugging using standard Python tools like pdb.

4. Deployment:

TensorFlow: Strong production deployment tools (TensorFlow Serving, TensorFlow Lite, TFX)

PyTorch: Historically weaker in production (improving with TorchServe, TorchScript

when to use tensor flow and pytorch;

TensorFlow: Best for production-ready systems, mobile/edge deployment, or when using Google Cloud tools.

PyTorch: Best for research, prototyping, and projects where flexibility and quick iteration matter most.

QUESTION 2: Jupyter Notebooks Use Cases in AI Development

Use Case 1: Exploratory Data Analysis and Model Prototyping

Data Exploration: Quickly visualize datasets, check distributions, identify outliers

Iterative Model Development: Test different architectures, hyperparameters, and see results immediately

Example: Loading a dataset, creating visualizations, trying different preprocessing techniques, and training multiple small models to gauge performance - all in an interactive environment.

Use Case 2: Educational Demonstrations and Documentation

Teaching Concepts: Combine explanatory text, mathematical equations, and executable code to explain AI concepts

Reproducible Research: Share complete analysis pipelines with results embedded

Example: Creating a tutorial that explains attention mechanisms in transformers with runnable code examples and visualizations of attention weights.

QUESTION3: How spaCy Enhances NLP vs Basic Python String Operations

Basic Python string methods (split(), replace(), find()) treat text as raw characters with no linguistic awareness. spaCy, on the other hand, provides a linguistically informed NLP pipeline

Tokenization & Sentence Segmentation: Splits text into words/sentences with awareness of punctuation, abbreviations, etc.

Part-of-Speech Tagging & Dependency Parsing: Identifies grammatical roles and syntactic relationships.

Named Entity Recognition (NER): Extracts entities like people, places, organizations, dates.

Lemmatization: Reduces words to their base form (e.g., running → run).

Efficiency: Built in Cython, spaCy processes millions of words quickly—far faster than naive Python loops.

QUESTION 3:

Target Applications

Scikit-learn;

Focused on classical machine learning: regression, classification, clustering, dimensionality reduction, and preprocessing.

Great for small to medium-sized datasets and traditional statistical models.

Not designed for deep learning or large-scale neural networks.

TensorFlow;

Built for deep learning and large-scale neural networks.

Supports advanced architectures (CNNs, RNNs, Transformers) and distributed training.

Includes deployment tools for mobile, web, and production environments.

Ease of Use for Beginners

Scikit-learn;

Very beginner-friendly with a *consistent API* (fit(), predict(), transform()).

Minimal boilerplate code—ideal for quickly testing algorithms and pipelines.

Excellent documentation and simple integration with NumPy, Pandas, and Matplotlib.

TensorFlow;

Steeper learning curve, especially for those new to deep learning.

High-level APIs like *Keras* make it easier, but still more complex than Scikit-learn.

Requires understanding of tensors, computational graphs, and GPU acceleration.

Community Support

Scikit-learn;

Mature, stable, and widely used in academia and industry for classical ML.

Strong documentation, tutorials, and a large base of contributors.

Slower pace of change—focused on reliability rather than cutting-edge features.

TensorFlow;

Backed by Google with a massive global community.

Rich ecosystem (TensorFlow Hub, TensorBoard, TensorFlow Lite, TensorFlow.js).

Rapidly evolving, with strong support for state-of-the-art deep learning research and production deployment.

CLASSICAL ML WITH SCIKIT-LEARN

```
First 5 rows of the dataset:
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa

Missing values in each column:
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64

Species encoding:
setosa -> 0
versicolor -> 1
virginica -> 2

Features shape: (150, 4)
Target shape: (150,)

Training set shape: (120, 4) (120,)
Testing set shape: (30, 4) (30,)

Model trained successfully.
Predictions made on test set.

Accuracy: 0.93
Precision (macro): 0.93
Recall (macro): 0.93

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
```

DEEPLEARNING WITH TENSORFLOW

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 69s 32ms/step - accuracy: 0.9100 - loss: 0.2950 - val_accuracy: 0.9847 - val_loss: 0.0465
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 76s 32ms/step - accuracy: 0.9863 - loss: 0.0438 - val_accuracy: 0.9882 - val_loss: 0.0353
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 58s 31ms/step - accuracy: 0.9907 - loss: 0.0279 - val_accuracy: 0.9906 - val_loss: 0.0330
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 84s 32ms/step - accuracy: 0.9940 - loss: 0.0186 - val_accuracy: 0.9877 - val_loss: 0.0354
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 57s 31ms/step - accuracy: 0.9952 - loss: 0.0151 - val_accuracy: 0.9875 - val_loss: 0.0407
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 59s 31ms/step - accuracy: 0.9965 - loss: 0.0101 - val_accuracy: 0.9900 - val_loss: 0.0351
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 57s 31ms/step - accuracy: 0.9968 - loss: 0.0081 - val_accuracy: 0.9904 - val_loss: 0.0374
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 58s 31ms/step - accuracy: 0.9975 - loss: 0.0084 - val_accuracy: 0.9907 - val_loss: 0.0356
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 58s 31ms/step - accuracy: 0.9981 - loss: 0.0057 - val_accuracy: 0.9907 - val_loss: 0.0393
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 61s 33ms/step - accuracy: 0.9978 - loss: 0.0058 - val_accuracy: 0.9921 - val_loss: 0.0381
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.9920 - loss: 0.0386
Test accuracy: 0.9921
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 122ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
```

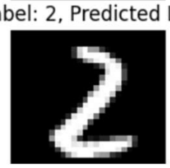
True Label: 2, Predicted Label: 2

True Label: 8, Predicted Label: 8

True Label: 2, Predicted Label: 2

True Label: 5, Predicted Label: 5

True Label: 2, Predicted Label: 2



NLP WITH SPACY

```
print("\n--- Extracted Entities ---")
for ent in doc.ents:
    if ent.label_ in ['PRODUCT', 'ORG']:
        print(f"Entity: {ent.text}, Type: {ent.label_}")

print("\n--- Sentiment Analysis Results ---")
print(f"Overall sentiment: {sentiment}")
print(f"Sentiment score: {sentiment_score}")
```

```
-- Extracted Entities ---
ntity: Samsung, Type: ORG
ntity: Apple, Type: ORG

-- Sentiment Analysis Results ---
verall sentiment: positive
entiment score: 4
```