Q1: Primary differences between TensorFlow and PyTorch. When to choose one over the other?

Differences:

Graph Execution: TensorFlow historically used static computation graphs (define-then-run), while PyTorch uses dynamic graphs (define-by-run), allowing real-time graph modifications.

API Design: TensorFlow (via Keras) offers high-level abstraction and production-ready tools (e.g., TF Serving). PyTorch provides a more Pythonic, imperative coding style, favored for research.

Debugging: PyTorch's dynamic nature simplifies debugging (standard Python tools). TensorFlow's static graphs require specialized tools (e.g., TF Debugger).

Deployment: TensorFlow excels in production (mobile/edge via TF Lite, web via TF.js). PyTorch relies on TorchServe or conversion (e.g., to ONNX) for deployment.

When to choose:

TensorFlow: Production pipelines (TFX), scalable deployment, or when leveraging Google Cloud AI services.

PyTorch: Research prototyping, dynamic models (e.g., variable-length sequences in NLP), or when Pythonic flexibility is prioritized.

Q2: Two use cases for Jupyter Notebooks in AI development

Exploratory Data Analysis (EDA): Interactively visualize datasets (e.g., using Matplotlib/Pandas), test preprocessing steps, and document insights alongside code.

Model Experimentation & Teaching: Iteratively train/test models (e.g., tweaking hyperparameters), visualize outputs (e.g., confusion matrices), and share executable tutorials with stakeholders.

Q3: How spaCy enhances NLP tasks vs. basic Python string operations?

Linguistic Intelligence: spaCy provides pre-trained models for POS tagging, NER, and dependency parsing, which require complex rules/regex in basic string operations.

Efficiency & Scalability: Optimized in Cython for fast batch processing (e.g., tokenizing 10K+ documents/sec), unlike Python's slower native string methods.

Context Awareness: Handles edge cases (e.g., "U.K." as one token) and linguistic nuances (lemmatization: "running" → "run"), while string operations (e.g., split()) fail on context.

2. Comparative Analysis: Scikit-learn vs. TensorFlow

Aspect	Scikit-learn	TensorFlow
Target Applications	Classical ML: Regression, clustering, SVMs, ensemble methods (Random Forests). Best for small-to-medium tabular data.	Deep Learning (DL): Neural networks (CNNs, RNNs), large-scale data (images, text, sequences). Supports classical ML via Keras.
Ease of Use for Beginners	Low barrier: Consistent API (fit(), predict()), minimal setup, and extensive built-in examples.	Moderate-to-high barrier: Requires understanding tensors, computational graphs, and hardware (GPU) setup. Simplified via Keras but still complex for non-DL tasks.
Community Support	Large, mature community with exhaustive documentation. Ideal for learning ML fundamentals.	Massive industry-backed ecosystem (Google). Rich resources (TensorFlow Hub, tutorials), but DL-focused.

```
spaCy model 'en_core_web_sm' loaded successfully.
Creating a dummy file 'train.ft' for demonstration purposes.
Dummy file 'train.ft' created. Please replace its content with your actual review data or provide your own file.
Creating a dummy file 'test.ft' for demonstration purposes.
Dummy file 'test.ft' created. Please replace its content with your actual review data or provide your own file.
Successfully loaded 3 reviews from 'train.ft'.
Successfully loaded 3 reviews from 'test.ft'.

--- Analyzing Reviews from TRAIN Dataset (3 reviews): ---


--- Processing Review 1 (TRAIN): ---
'I absolutely love the new Apple iPhone 15! The camera is incredible, and the battery life is surprisingly good. Highly recommen
d this amazing product.'
--- Extracted Named Entities (Products & Brands): ---
  Entity: 'Apple' (Type: ORG)
  Entity: 'iPhone 15' (Type: PRODUCT)

--- Sentiment Analysis (Rule-based): ---
  Positive keywords found: 5
  Negative keywords found: 1
  Overall Sentiment: Positive

--- Processing Review 2 (TRAIN): ---
'The Samsung Galaxy S23 Ultra has an excellent display, but the software updates are a bit slow.'
--- Extracted Named Entities (Products & Brands): ---
  No common product/brand entities found by spaCy's default NER for this review.

--- Sentiment Analysis (Rule-based): ---
  Positive keywords found: 1
  Negative keywords found: 1
  Overall Sentiment: Neutral
```

```
CNN Model Architecture:
CNN(
  (conv1): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (relu1): ReLU()
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (relu2): ReLU()
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc): Linear(in_features=1568, out_features=10, bias=True)
)

Starting model training...
Epoch [1/5], Step [100/938], Loss: 0.2416
Epoch [1/5], Step [200/938], Loss: 0.0791
Epoch [1/5], Step [300/938], Loss: 0.0647
Epoch [1/5], Step [400/938], Loss: 0.3918
Epoch [1/5], Step [500/938], Loss: 0.0545
Epoch [1/5], Step [600/938], Loss: 0.0610
Epoch [1/5], Step [700/938], Loss: 0.0288
Epoch [1/5], Step [800/938], Loss: 0.0529
Epoch [1/5], Step [900/938], Loss: 0.0771
Epoch [2/5], Step [100/938], Loss: 0.0195
Epoch [2/5], Step [200/938], Loss: 0.0936
Epoch [2/5], Step [300/938], Loss: 0.0051
Epoch [2/5], Step [400/938], Loss: 0.1629
Epoch [2/5], Step [500/938], Loss: 0.0263
Epoch [2/5], Step [600/938], Loss: 0.0263
Epoch [2/5], Step [700/938], Loss: 0.0138
```

```
Missing values per column:
 |               | 0   |
|:--------------|:----|
| Id            | 0   |
| SepalLengthCm | 0   |
| SepalWidthCm  | 0   |
| PetalLengthCm | 0   |
| PetalWidthCm  | 0   |
| Species       | 0   |

Encoded Species labels (first 5):
 [0 0 0 0 0]
Original Species to Encoded Mapping:
 [('Iris-setosa', np.int64(0)), ('Iris-versicolor', np.int64(1)), ('Iris-virginica', np.int64(2))]

Decision Tree Classifier trained successfully.

Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
```

```python
# Buggy MNIST TensorFlow Code
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.losses import MeanSquaredError  # Wrong loss

# Load MNIST data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize
x_train = x_train / 255.0
x_test = x_test / 255.0

# Bug: y_train is not one-hot encoded but using wrong loss
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')  # Multi-class classification
])

model.compile(optimizer='adam',
              loss=MeanSquaredError(),  # ❌ Wrong loss for classification
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
```

```
Epoch 1/5
```

```
Epoch 1/5
1875/1875 [==============================] - 13s 6ms/step - loss: 27.3046 - accuracy: 0.0986 - val
0.1023
Epoch 2/5
1875/1875 [==============================] - 8s 4ms/step - loss: 27.3046 - accuracy: 0.1007 - val_
0.0981
Epoch 3/5
1875/1875 [==============================] - 8s 5ms/step - loss: 27.3046 - accuracy: 0.1009 - val_
0.0981
Epoch 4/5
1875/1875 [==============================] - 9s 5ms/step - loss: 27.3046 - accuracy: 0.0995 - val_
0.0985
Epoch 5/5
1875/1875 [==============================] - 8s 4ms/step - loss: 27.3046 - accuracy: 0.1013 - val_
0.0982
```

[23]: <keras.callbacks.History at 0x7f9bd52f6590>

[24]:

```
# Fixed MNIST TensorFlow Code
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical

# Load MNIST data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize
x_train = x_train / 255.0
x_test = x_test / 255.0
```