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

Aspect	TensorFlow	PyTorch
Computation Graph	Uses static computation graphs (though TF 2.x allows eager execution).	Uses dynamic computation graphs (define-by-run).
Ease of Debugging	Harder to debug due to static graphs (before TF 2.x).	Easier to debug because graphs are built on the fly, similar to normal Python code.
Deployment	Better for production/deployment (TensorFlow Serving, TensorFlow Lite, TensorFlow.js).	Deployment is possible (TorchServe, ONNX), but less mature.
Community & Ecosystem	Larger ecosystem for production, mobile, and cross-platform.	Strong research community; often the first to receive cutting-edge models.

When to choose:

- **PyTorch:** Research, prototyping, experimental models, or when you need flexibility in graph execution.
- **TensorFlow:** Production-ready applications, deployment across devices, and integration into large-scale pipelines.

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

1. **Exploratory Data Analysis (EDA):**
    - Visualizing data distributions, correlations, missing values.
    - Iterative testing of preprocessing steps with immediate feedback.
2. **Model Prototyping and Documentation:**
    - Quickly build, train, and evaluate ML/DL models.
    - Combine code, visualizations, and markdown explanations for reproducibility and sharing with teams.

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

- **Pre-built NLP Pipelines:** Tokenization, lemmatization, POS tagging, named entity recognition (NER), and dependency parsing.
- **Speed & Accuracy:** Optimized Cython backend for large-scale text, faster than naive Python string operations.

- **Consistency & Reliability:** Handles edge cases (punctuation, contractions, multilingual support) that simple string operations often fail to address.
- **Integration:** Easy interoperability with ML pipelines for downstream tasks like sentiment analysis or text classification.

2. Comparative Analysis: Scikit-learn vs. TensorFlow

Feature	Scikit-learn	TensorFlow
Target Applications	Classical ML: regression, classification, clustering, feature engineering.	Deep learning: CNNs, RNNs, Transformers, large-scale neural networks.
Ease of Use for Beginners	Very beginner-friendly: simple API, minimal setup, concise syntax.	Steeper learning curve, especially for beginners in computational graphs and tensors.
Community Support	Mature, large community; many tutorials for classical ML.	Very active research and industry community; extensive documentation and examples for deep learning.

Summary:

- Use **Scikit-learn** for small-to-medium datasets, classical ML, and rapid prototyping.
- Use **TensorFlow** for complex neural networks, large datasets, and production-ready deployment pipelines.

PART 3

1. Ethical Considerations

When working with **MNIST** (handwritten digits) or **Amazon Reviews** (text data), potential biases can arise:

Potential Biases

MNIST:

- **Digit representation bias**: MNIST mostly has digits written by American students in the 1990s. Digits from other regions may be written differently, so the model may misclassify.
- **Class imbalance**: If some digits appear more frequently than others, the model may favor the more common digits.

Amazon Reviews:

- **Sentiment bias**: Reviews may have cultural or product-category biases. For example, some products may receive more positive reviews because of brand popularity rather than quality.

- **Demographic bias**: Language used in reviews may reflect age, gender, or cultural background, influencing sentiment detection unfairly.
- **Neglecting sarcasm or context**: Rule-based systems may misclassify sarcastic reviews as positive.

Mitigating Biases

Tools like TensorFlow Fairness Indicators:

- Can **measure fairness** across groups (e.g., gender, location, or review category).
- Helps **identify if the model treats some groups unfairly**.
- Example: Check if positive/negative sentiment predictions are balanced across product categories.

spaCy's rule-based systems:

- Can **preprocess text to normalize language**, detect and adjust for bias patterns.
- For example:
    - Remove words that reflect demographic bias.
    - Add rules to detect sarcasm or negation in sentiment analysis.

Summary:
The key is **awareness and monitoring—**use fairness tools to identify bias and NLP rules to reduce it before model training.

2. Troubleshooting Challenge

Imagine a **buggy TensorFlow script** like this:

```
import tensorflow as tf
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Dense(64, activation='relu', input_shape=(28,28)),
    layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',  # ❌ wrong loss
              metrics=['accuracy'])

# x_train shape: (60000,28,28)
# y_train shape: (60000,)  # labels 0-9
```

Problems:

1. **Input shape mismatch**: Dense layer expects 1D input, but MNIST images are `(28,28)`.
2. **Wrong loss function**: `binary_crossentropy` is for 2-class problems, not 10 classes.
3. **y_train shape**: Needs one-hot encoding for `categorical_crossentropy`.

Fixed Code:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.utils import to_categorical

# Flatten input and one-hot encode labels
x_train = x_train.reshape(-1, 28*28).astype('float32') / 255
y_train = to_categorical(y_train, 10)

x_test = x_test.reshape(-1, 28*28).astype('float32') / 255
y_test = to_categorical(y_test, 10)

model = models.Sequential([
    layers.Dense(64, activation='relu', input_shape=(28*28,)),
    layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5, batch_size=32, validation_data=(x_test,
y_test))
```

Changes made:

- Flattened input `(28*28,)`.
- Changed loss to `categorical_crossentropy`.
- One-hot encoded labels.
- Normalized pixel values.