1. Short Answer Questions

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

Feature	PyTorch	TensorFlow
Computation Graph	**Dynamic (Define-by-Run)**	**Static (Define-and-Run)** (Though TF 2.x defaults to Eager Execution, the static graph mode is still used for production optimization via tf.function).
Debugging	Easier, as it is more **"Pythonic."** Standard Python debuggers (like pdb) can be used directly, and execution is step-by-step.	Historically harder, but **Eager Execution** in TF 2.x made it much simpler. Debugging in graph mode (tf.function) can still be complex.
Deployment	Used **TorchServe** and **ONNX** (Open Neural Network Exchange). Historically considered catching up in terms of comprehensive deployment ecosystem.	Excellent and mature ecosystem with robust tools like **TensorFlow Serving**, **TensorFlow Lite (TFLite)** for mobile/edge, and **TensorFlow.js** for web.
Model Building API	More **low-level, object-oriented** (using nn.Module), giving greater control and flexibility.	Excellent high-level API through **Keras** (now built-in as tf.keras), which is easier and faster for standard models.

When to Choose Which:

- **Choose PyTorch when:**
    - **Research & Prototyping:** The dynamic graph and flexible, intuitive Python-like structure make it ideal for quick experimentation, cutting-edge research, and rapid iteration on new model architectures.[1]
    - **Custom Models:** You need intricate, step-by-step control over the training loop or are dealing with models with dynamic sequence lengths (e.g., complex NLP tasks).
- **Choose TensorFlow when:**
    - **Production Deployment:** Your primary focus is taking the model to large-scale production, especially across diverse platforms like mobile, web, and enterprise servers (due to TF Serving/TFLite/TF.js).[2]

- **Enterprise Scale:** You need a mature, comprehensive MLOps ecosystem (like TFX) and robust cross-platform compatibility, backed heavily by Google.[3]
- **Ease of Use for Standard Models:** Using Keras (tf.keras) makes it exceptionally easy to build and train standard neural networks.[4]

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

1. **Exploratory Data Analysis (EDA) and Visualization:**
    - **Description:** Jupyter Notebooks allow developers to load datasets and process them cell-by-cell.[5] This interactive environment is perfect for visualizing data distributions, checking for missing values, performing feature engineering, and instantly plotting charts (using libraries like Matplotlib or Seaborn).
    - **Value:** It creates a transparent, human-readable record of data cleaning and initial insights, ensuring data integrity and reproducibility before any model training begins.
2. **Machine Learning Prototyping and Experimentation:**
    - **Description:** An AI engineer can develop a model from scratch—defining the architecture, training the model for a few epochs, and immediately evaluating the performance—all within sequential cells. If the results are poor, they can go back to an earlier cell, change a hyperparameter, and re-run the training process quickly.
    - **Value:** This interactive workflow is crucial for **rapid iteration** and debugging.[6] The notebook acts as a living document of the entire experiment, combining the runnable code, output metrics, and explanatory text (Markdown) in one file.[7]

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

Basic Python string operations (like .split(), .lower(), and regex) only manipulate text at a character or surface-level (lexical) level, treating it as a sequence of symbols.

spaCy enhances NLP tasks by providing **linguistic intelligence** and **contextual understanding**, which basic string operations cannot:[8]

1. **Linguistic Annotation and Tokenization:**
    - **Basic String Ops:** .split(' ') simply breaks text by whitespace, failing on punctuation (e.g., "U.S.A." becomes three separate tokens).
    - **spaCy:** It uses a linguistically-informed **Tokenizer** to correctly segment text into meaningful tokens (words, punctuation, abbreviations), and then provides rich attributes for each token, such as **Part-of-Speech (POS) tagging** (e.g., *noun, verb*), **Lemmatization** (reducing a word to its base form, e.g., *running* to *run*), and **Morphology**.[9]
2. **Information Extraction:**
    - **Basic String Ops:** Requires complex, brittle regular expressions to find patterns like phone numbers or dates, with no ability to understand context.
    - **spaCy:** It includes a highly optimized, pre-trained component for **Named Entity Recognition (NER)**. It can automatically identify, classify, and label entities in the text (e.g., *PERSON, ORG, GPE, DATE*) with high accuracy, which is essential for tasks like information retrieval and document summarization.
3. **Syntactic Understanding:**
    - **Basic String Ops:** Has no concept of grammar or word relationships.
    - **spaCy:** It performs **Dependency Parsing**, which reveals the grammatical relationships between words (who is doing what to whom). This structure is critical for advanced tasks like building knowledge graphs or determining the subject of a sentence.

---

2. Comparative Analysis: Scikit-learn vs. TensorFlow

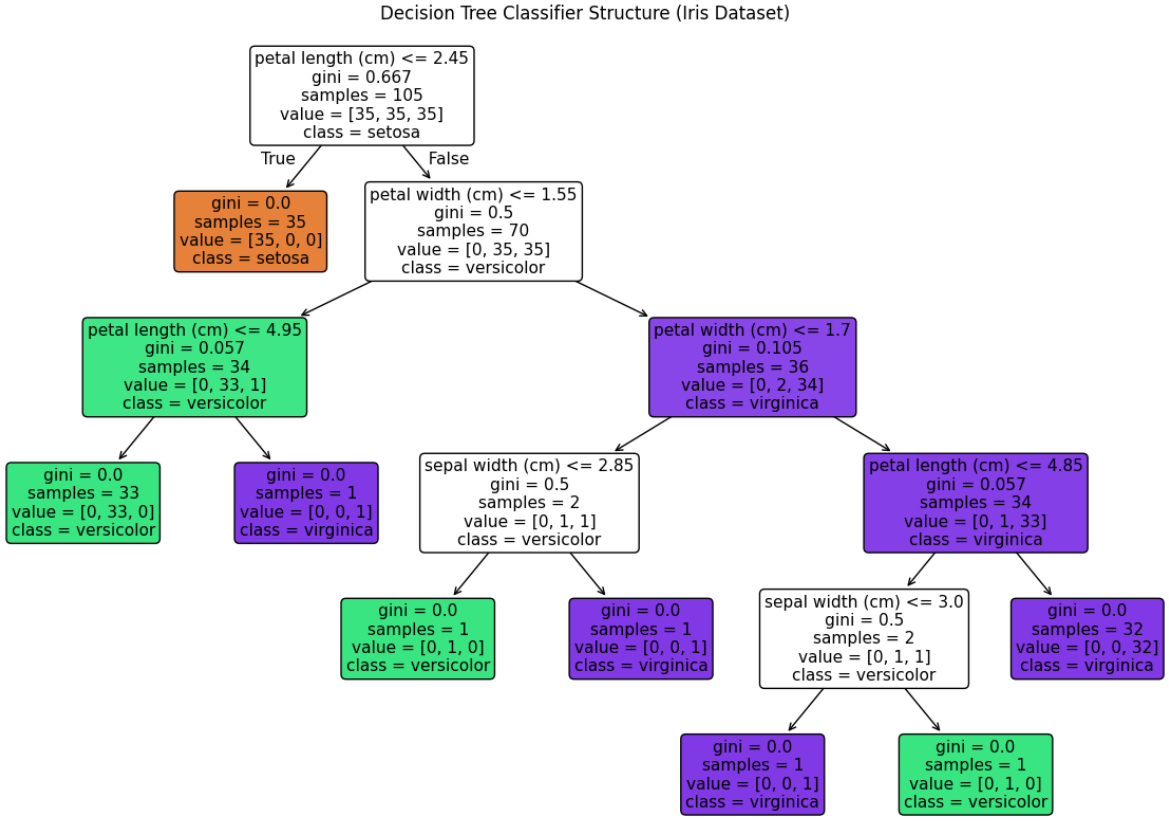
Criterion	Scikit-learn (often written as sklearn)	TensorFlow
Target Applications	**Classical Machine Learning (ML)**: Used for traditional algorithms like Logistic Regression, Random Forest, Support Vector Machines (SVM), K-Means Clustering, and dimensionality reduction (PCA). Primarily works with **structured (tabular) data**.	**Deep Learning (DL)**: Used for building complex **Neural Networks** (NNs) like Convolutional NNs (CNNs), Recurrent NNs (RNNs), and Transformers. Designed for **unstructured data** (images, text, audio, video).

Ease of Use for Beginners	**High.** Very easy to use due to its consistent, unified API (the "fit-predict-transform" interface). Excellent for getting started with basic ML models quickly.	**Moderate to High (via Keras).** While historically steeper, the integration of **Keras** (tf.keras) provides a very simple, high-level API for model definition, making it accessible to beginners for standard deep learning tasks. Custom deep learning still requires more technical setup.
Community Support	**Very Strong.** It is the industry standard for classical ML, with extensive official documentation, academic usage, and robust support, though its focus remains fixed on the non-deep learning algorithms.	**Massive and Active.** Backed by Google, it has an enormous global community of researchers, developers, and practitioners. It is the dominant choice for cutting-edge deep learning research, offering countless tutorials, pre-trained models, and MLOps tools.

MODEL OUTPUT IMAGES

Classical Ml with Scit-learn

```
--- 1. Data Loading ---
Features: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target Classes: ['setosa' 'versicolor' 'virginica']
------------------------------
--- 2. Data Preprocessing ---
Missing values check: No missing values found (Standard Iris dataset is clean).
Label Encoding: Already handled (Species are encoded as 0, 1, 2).
Training set size: 105 samples
Testing set size: 45 samples
------------------------------
--- 3. Model Training ---
Decision Tree Classifier training complete.
------------------------------
--- 4. Model Prediction and Evaluation ---
1. Accuracy Score: 0.9333

2. Classification Report (Precision, Recall, F1-Score):
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       1.00      0.80      0.89        15
   virginica       0.83      1.00      0.91        15

    accuracy                           0.93        45
   macro avg       0.94      0.93      0.93        45
weighted avg       0.94      0.93      0.93        45

3. Confusion Matrix:
            setosa  versicolor  virginica
setosa          15           0          0
versicolor       0          12          3
virginica        0           0         15
------------------------------
```


Decision Tree Classifier Structure (Iris Dataset)
petal length (cm) <= 2.45
gini = 0.667
samples = 105
value = [35, 35, 35]
class = setosa
True
False
gini = 0.0
samples = 35
value = [35, 0, 0]
class = setosa
petal width (cm) <= 1.55
gini = 0.5
samples = 70
value = [0, 35, 35]
class = versicolor
petal length (cm) <= 4.95
gini = 0.057
samples = 34
value = [0, 33, 1]
class = versicolor
petal width (cm) <= 1.7
gini = 0.105
samples = 36
value = [0, 2, 34]
class = virginica
gini = 0.0
samples = 33
value = [0, 33, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
sepal width (cm) <= 2.85
gini = 0.5
samples = 2
value = [0, 1, 1]
class = versicolor
petal length (cm) <= 4.85
gini = 0.057
samples = 34
value = [0, 1, 33]
class = virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
sepal width (cm) <= 3.0
gini = 0.5
samples = 2
value = [0, 1, 1]
class = versicolor
gini = 0.0
samples = 32
value = [0, 0, 32]
class = virginica
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor

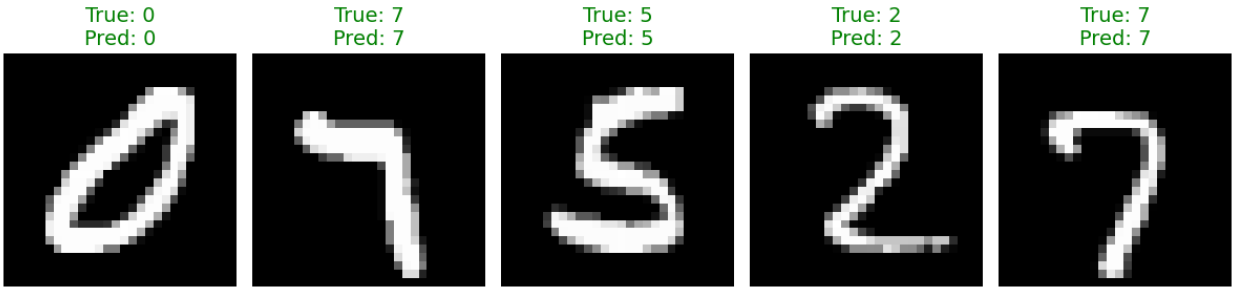
Deep learning with Tensorflow

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
dropout (Dropout)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 128)	401,536
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

```
Total params: 421,642 (1.61 MB)

Trainable params: 421,642 (1.61 MB)

Non-trainable params: 0 (0.00 B)

Training model (this may take a few minutes)...
Epoch 1/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 24s 50ms/step - accuracy: 0.9021 - loss: 0.3170 - val_accuracy: 0.9838 - val_loss: 0.0590
Epoch 2/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 21s 49ms/step - accuracy: 0.9684 - loss: 0.1033 - val_accuracy: 0.9883 - val_loss: 0.0446
Epoch 3/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 23s 54ms/step - accuracy: 0.9765 - loss: 0.0771 - val_accuracy: 0.9902 - val_loss: 0.0375
Epoch 4/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 25s 60ms/step - accuracy: 0.9804 - loss: 0.0661 - val_accuracy: 0.9902 - val_loss: 0.0360
Epoch 5/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 22s 51ms/step - accuracy: 0.9820 - loss: 0.0565 - val_accuracy: 0.9903 - val_loss: 0.0335

Evaluating model performance...
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Test Accuracy: 99.14%
--- SUCCESS: Test accuracy is above the 95% goal! ---

Visualizing 5 sample predictions...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 168ms/step
```

Model Predictions on 5 Sample Test Images



NLP with spaCy

```
spaCy model 'en_core_web_sm' loaded successfully.
--------------------------------------------------
PRODUCT REVIEW NLP ANALYSIS
--------------------------------------------------

Review 1:
  Text: I love the new Sonic X headphones. The battery life lasts forever and the sound quality is amazing. Highly recommend this product from TechCor
p.
  --- NER Results ---
  Extracted Entities: TechCorp, X, Highly, I, The, Sonic
  --- Sentiment Analysis (Rule-Based) ---
  Sentiment: Positive (Pos Score: 3, Neg Score: 0)

Review 2:
  Text: This Ultra-Wipe Cleaner arrived broken and leaked everywhere. The packaging was terrible. I am extremely disappointed and will return it.
  --- NER Results ---
  Extracted Entities: Wipe, Ultra, I, This, Cleaner, The
  --- Sentiment Analysis (Rule-Based) ---
  Sentiment: Negative (Pos Score: 0, Neg Score: 5)

Review 3:
  Text: The Cozy Blanket is the best purchase of the year! It's super soft and the crimson color is vibrant. A truly wonderful experience.
  --- NER Results ---
  Extracted Entities: The Cozy Blanket, It, A, Blanket, Cozy, The
  --- Sentiment Analysis (Rule-Based) ---
  Sentiment: Positive (Pos Score: 4, Neg Score: 0)

Review 4:
  Text: My delivery of the Alpha Laptop was delayed by two weeks, which was frustrating. The laptop itself is fine, but the service was terrible.
  --- NER Results ---
  Extracted Entities: Laptop, The, Alpha, My
  --- Sentiment Analysis (Rule-Based) ---
  Sentiment: Negative (Pos Score: 0, Neg Score: 3)

Review 5:
  Text: Excellent value for the Stellar Mixer. It makes baking so much easier. No complaints about the quality or performance.
  --- NER Results ---
  Extracted Entities: Mixer, It, No, Stellar, Excellent
  --- Sentiment Analysis (Rule-Based) ---
  Sentiment: Positive (Pos Score: 2, Neg Score: 0)

--------------------------------------------------
```

ETHICAL CONSIDERATIONS

Potential Biases in the Amazon Reviews Model

- **Lexical/Semantic Bias**
    - **Description & Example:** The model's sentiment is **entirely dependent on fixed, pre-defined lists** (e.g., POSITIVE_WORDS, NEGATIVE_WORDS).
    - **Impact on Model:**
        - It fails to understand **negation** (e.g., "The battery life is not terrible" is scored as Negative due to "terrible").
        - It fails to understand **sarcasm** ("The delay was a wonderful experience.").
        - It is biased towards simple, direct language.

- **Domain & Context Bias**
    - **Description & Example:** The lexicon is **generic**. Words have different sentiment polarity across domains.
    - **Impact on Model:**
        - A review stating the **"laptop is heavy"** should be negative, but if the word "heavy" isn't in the negative list, the model is biased toward misclassification.
        - It lacks the ability to analyze sentiment based on the **product feature** being discussed.
- **Socio-Linguistic Bias**
    - **Description & Example:** The word lists are typically based on **standard English word usage**.
    - **Impact on Model:**
        - The model is **biased against non-standard or regional English dialects, slang, or colloquialisms** where positive or negative sentiment might be expressed using terms not included in the generic lexicon.

Mitigation strategies

A. Mitigation with TensorFlow Fairness Indicators (For ML/Deep Learning Models)

Fairness Indicators is designed to **audit and quantify** bias in *trained machine learning models* (like an actual neural network-based sentiment classifier or your MNIST classifier).

- **Audit and Quantify:** You would use Fairness Indicators to compute metrics (like **False Positive Rate** or **Accuracy**) across different pre-defined **slices** of your data (e.g., reviews containing known geo-political terms or names associated with a specific group).
- **Identify Disparities:** If the False Positive Rate (e.g., flagging a positive review as negative) is significantly higher for reviews written by users in one geographic region (based on location/language metadata) compared to others, a bias is confirmed.

- **Remediate (Post-Audit):** While Fairness Indicators itself is an evaluation tool, it points to the need for mitigation strategies like **MinDiff**. MinDiff is an optimization technique used during retraining to penalize models that show performance differences across defined slices, thereby reducing the observed bias.

B. Mitigation with spaCy's Rule-Based Systems (For your Current Script)

Because your current system is rule-based, mitigation happens via **adding more granular linguistic rules** to increase precision and transparency:

- **Negation Handling:** You can add a Matcher pattern to specifically look for negative keywords (like "not," "no," "never") immediately preceding a positive word. If this pattern is matched, the rule should **flip the sentiment score** for that token from positive to negative.
- **Contextual Rules (POS/Dependency):** Implement a rule to ensure sentiment words are modifying relevant nouns. For example, check if "terrible" is directly linked to the product noun ("laptop") or a secondary entity ("customer service"). This helps distinguish product quality issues from service issues.
- **Domain Adaptation:** Create **Product-Specific Lexicons**. By loading a separate set of words associated with a product category (e.g., for "Electronics," add "fast," "laggy"), you can override or supplement the generic lexicon based on the identified product entity (which your current NER component finds).