Part 1: Theoretical Understanding
1. Short Answer Questions

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

Programming Style & Ease of Use:

- **TensorFlow historically used a "define-then-run" paradigm**, where you first build a static computational graph and then execute it within a `tf.Session`. While TensorFlow 2.0 adopted eager execution by default, it still heavily promotes `tf.function` to create graphs for performance, which can sometimes be less intuitive for debugging.

Graph Construction:

- **PyTorch has a dynamic computation graph.**The graph is built on-the-fly during the forward pass. This is a major advantage for models with variable-length inputs (e.g., RNNs on text) or structures that change during execution.

- **TensorFlow uses a static computation graph** (when using `tf.function`). The graph is defined once and then executed. This allows for advanced optimizations and deployment to non-Python environments (like mobile, embedded) more easily.

API and Ecosystem:

- **TensorFlow** offers a more monolithic but complete ecosystem** with tools like **TensorFlow Serving(for deployment), TensorFlow Lite(for mobile), and TensorFlow.js (for web). It has strong production-grade deployment pipelines.
- **PyTorch has a more modular ecosystem.** Its core is lean, with libraries like TorchServe(for serving) and TorchMobile evolving rapidly. It is tightly integrated with the Hugging Face `transformers` library, making it a favorite in NLP research.

When to Choose One Over the Other:**

PyTorch
1. Effective for research or when you're working on prototyping.The dynamic graph and intuitive debugging make iterative development faster
2. Your work is primarily in academia or novel model development, especially in NLP, where the community heavily favors PyTorch.
3. You prioritize code readability and a 'Pythonic' programming style.

TensorFlow
1. Your primary goal is deploying models to production at scale, especially on mobile or embedded devices (TensorFlow Lite) or using web browsers (TensorFlow.js).
2. You are working in an enterprise environment where a stable, mature, and end-to-end production pipeline is critical.
3. You are focused on applications like large-scale image classification, recommendation systems, or other established deep learning tasks** where TensorFlow's ecosystem is very strong.

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

1. Exploratory Data Analysis (EDA) and Rapid Prototyping:

Jupyter Notebooks are ideal for the initial stages of an AI project. A data scientist can load a data set, run statistical summaries, and create visualizations (e.g., histograms, scatter plots) in an interactive, cell-by-cell manner.
This allows for immediate feedback and hypothesis testing about the data's distribution, missing values, and feature correlations before committing to a specific model architecture. It's the perfect environment for quickly testing a small idea or a new algorithm without the overhead of writing a full script.

2. Interactive Model Training and Visualization:

During model development, Jupyter Notebooks allow developers to train a model incrementally and visualize the results in real-time. For example, you can train a model for a few epochs in one cell, then execute another cell to plot the training and validation loss curves.
This interactivity helps in diagnosing issues like over fitting early on. Furthermore, for computer vision tasks, you can display sample images with model predictions or activation maps directly in the notebook, making it an excellent tool for debugging and presenting results to stakeholders.

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

Basic Python string operations (e.g., `str.split()`, `str.find()`, regular expressions) work on a superficial, character-level basis. spaCy provides a sophisticated, linguistic-aware processing pipeline that understands the structure and meaning of text.

Key enhancements include:

Linguistic Features:

spaCy performs ;
- ✓ **Part-of-Speech (POS) Tagging**(identifying nouns, verbs, etc.),
- ✓ **Dependency Parsing** (understanding grammatical relationships between

words)
- ✓ **Lemmatization** (reducing words to their base dictionary form, e.g., "was" -> "be"), which are impossible with simple string splits.

Statistical Models:
Instead of relying on hand-crafted rules, spaCy uses pre trained statistical models to make predictions about the text. This allows it to generalize well to real-world, messy language with slang, ambiguities, and varied sentence structures.

Named Entity Recognition (NER):
spaCy can identify and categorize real-world objects like persons, organizations, locations, dates, and monetary values. Identifying "Apple" as a company versus a fruit requires contextual understanding far beyond string matching.

Efficiency and Accuracy:
spaCy is designed for processing large volumes of text and is significantly more accurate and robust for complex NLP tasks than rule-based string operations, which become unimaginably complex for anything beyond simple patterns.

2. Comparative Analysis
Compare Scikit-learn and TensorFlow :

Feature	Scikit-learn	Tensor Flow
Target applications	Classical Machine Learning. It is the go to library for algorithms like linear regression, logistic regression, support vector machines, random forests, and k-Nearest Neighbors (k-NN). It excels at tasks on structured, tabular data	Deep Learning It is designed for building and training large-scale-neural networks such as Convolutional Neural Networks (CNNs) for image recognition, Recurrent Neural Networks (RNNs) for sequence data, and transformers for NLP.
Ease of use for beginners	**Very high** It features a clean, consistent, and well documented API. A beginner can train a powerful model with just a few lines of code without understanding the underlying mathematics in depth	**Moderate to steep** The learning curve is stepper. Beginners must understand concepts like tensors, computational graphs, automatic differentiation and GPU programming.

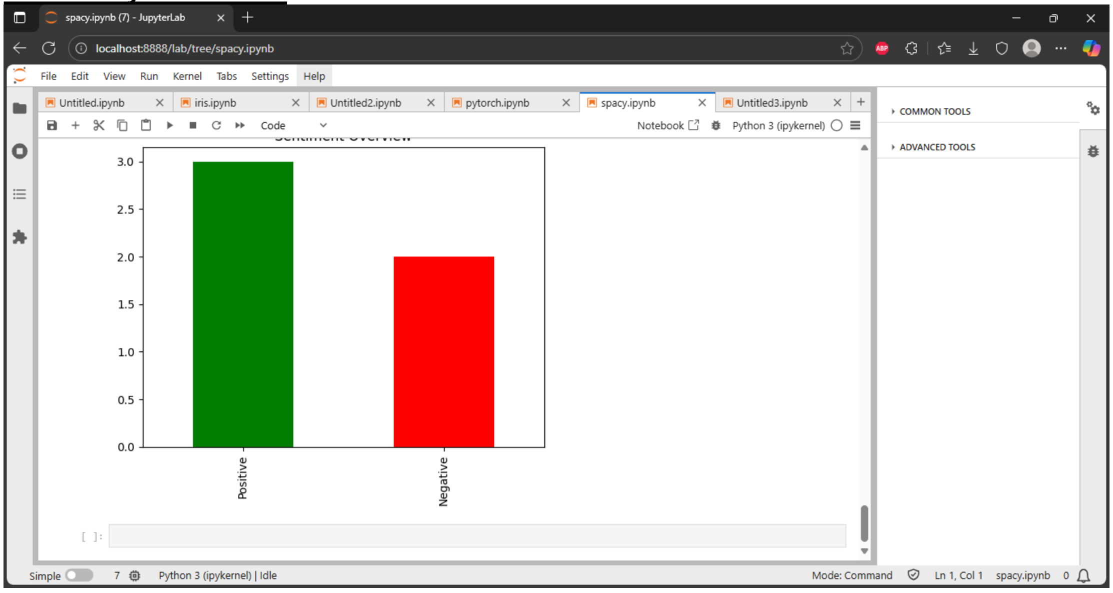
		While Keras simplifies the process, debugging deep learning models is inherently more complex
Community Support	**Extremely strong in the Data Science and traditional ML community**. It is one of the most established and well-maintained libraries, with a vast amount of tutorials. Examples on sites like Stack Overflow, and is a standard part of most ML curricula.	**Massive and diverse, focused on both research and industry.** It has one of the largest communities in AI, backed by Google. Support is extensive for both cutting edge research and production deployment

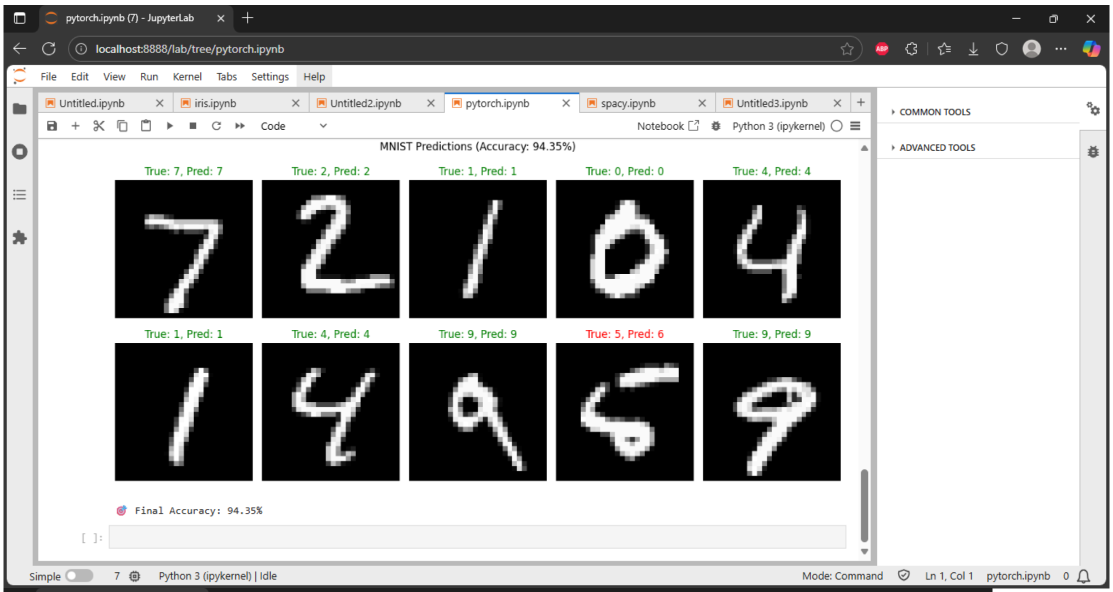
<u>Practical Implementation Task Results</u>

<u>Tasks</u>

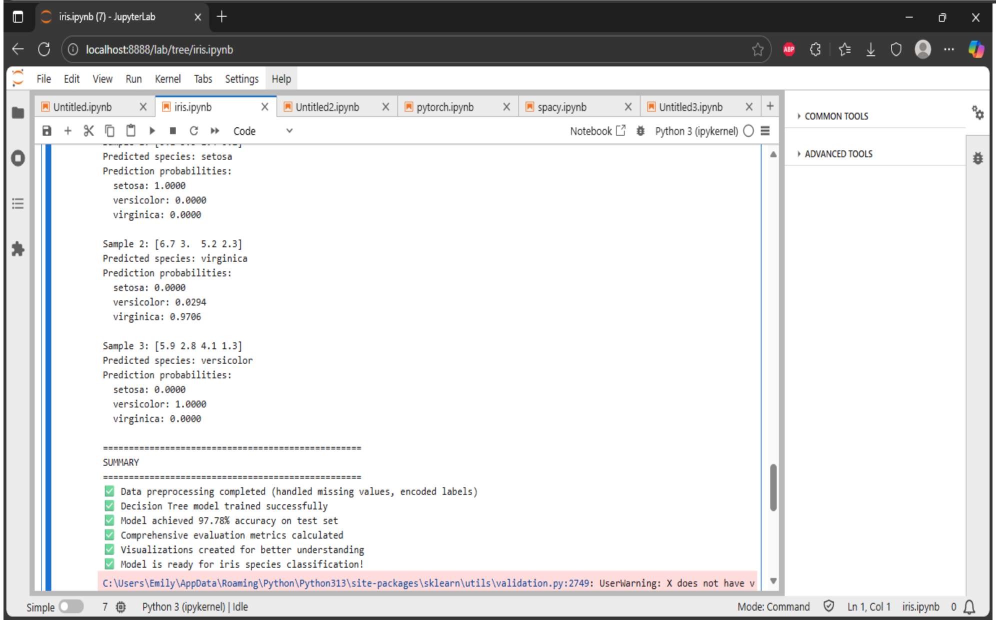
1. Classical ML with Scikit-learn
2. Deeplearning with Pytorch
3. NLP with spaCy

<u>Summary of results</u>





```
🎯 Final Accuracy: 94.35%
```



```
Predicted species: setosa
Prediction probabilities:
  setosa: 1.0000
  versicolor: 0.0000
  virginica: 0.0000

Sample 2: [6.7 3.  5.2 2.3]
Predicted species: virginica
Prediction probabilities:
  setosa: 0.0000
  versicolor: 0.0294
  virginica: 0.9706

Sample 3: [5.9 2.8 4.1 1.3]
Predicted species: versicolor
Prediction probabilities:
  setosa: 0.0000
  versicolor: 1.0000
  virginica: 0.0000

==================================================
SUMMARY
==================================================
✅ Data preprocessing completed (handled missing values, encoded labels)
✅ Decision Tree model trained successfully
✅ Model achieved 97.78% accuracy on test set
✅ Comprehensive evaluation metrics calculated
✅ Visualizations created for better understanding
✅ Model is ready for iris species classification!
C:\Users\Emily\AppData\Roaming\Python\Python313\site-packages\sklearn\utils\validation.py:2749: UserWarning: X does not have v
```

Part 3 Ethics and Optimization

Potential Biases Identified

a) *MNIST Digit Recognition Model*

Bias Source: MNIST was built from handwriting samples by mostly Western users, making it less inclusive of other scripts (e.g., Arabic, Hindi).

Impact: The model may mis-classify digits written in non-Latin styles.

Mitigation:

- Use a more **diverse dataset** (global handwriting samples).

- Apply **data augmentation** (rotation, scaling) to simulate handwriting variation.
- Evaluate subgroup performance using TensorFlow Fairness Indicators.

b) *Amazon Reviews NLP Model*

Bias Source: Language and tone variations — slang or sarcasm may be misinterpreted.

Impact: Sentiment polarity may not generalize across cultures or product types.

Mitigation:

- Balance review samples across product categories.
- Extend spaCy’s rule-based sentiment to account for negation and context.
- Regularly validate predictions for fairness across brand/product groups.

o