WEEK 3 ASSIGNMENT – GROUP MEMBERS

	Name	Email Address
1	Faith Mutua	charlesfaith157@gmail.com
2	Francis Gachoki	kranchezjb@gmail.com

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?

TensorFlow (developed by Google) uses a graph-based execution model, where computations are defined as a static graph (though it supports eager execution for dynamic graphs since version 2.0). It emphasizes production deployment with tools like TensorFlow Serving and TensorFlow Lite for mobile/edge devices. PyTorch (developed by Meta) is built around dynamic computation graphs (imperative programming), making it more intuitive for rapid prototyping and debugging, as code runs line-by-line like standard Python.

Key differences:

- **Graph Execution**: TensorFlow's static graphs optimize for performance in large-scale deployments but can be less flexible. PyTorch's dynamic graphs allow easier modifications during runtime.
- **API Style**: TensorFlow has a more declarative API (define-then-run), while PyTorch is imperative (define-by-run).
- **Ecosystem**: TensorFlow has stronger support for distributed training and production tools; PyTorch excels in research with seamless integration to libraries like Hugging Face.

Choose TensorFlow for production environments, scalability in enterprise (e.g., recommendation systems at scale), or when needing built-in deployment tools. Choose PyTorch for research, experimentation, or when flexibility and ease of debugging are priorities (e.g., custom model architectures in academia).

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

1. **Interactive Data Exploration and Prototyping**: Jupyter allows mixing code, text, and visualizations in one document. For AI, this is ideal for loading datasets (e.g., via pandas), preprocessing, training small models, and iterating quickly—e.g., experimenting with hyperparameters in a PyTorch model and plotting loss curves with matplotlib inline.
2. **Collaboration and Documentation**: Teams can share notebooks with embedded explanations, results, and code. In AI projects, this supports reproducible research, like documenting a TensorFlow pipeline for image classification, including model summaries, evaluation metrics, and visualizations, making it easy for others to review or extend.

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

Basic Python string operations (e.g., split(), find(), regex via re module) are low-level and require manual implementation for tasks like tokenization or entity extraction, often leading to inefficient, error-prone code for complex text.

spaCy enhances NLP by providing:

- **Pre-trained Pipelines**: Industrial-strength models for tokenization, part-of-speech tagging, dependency parsing, and named entity recognition (NER) out-of-the-box, trained on large corpora for accuracy.
- **Efficiency**: Optimized Cython-based implementation for speed on large texts, unlike slow pure-Python loops.
- **Extensibility**: Easy to add custom rules or integrate with ML models (e.g., for sentiment analysis).
- **Visualization**: Built-in tools like displacy for rendering parse trees or entities.

For example, extracting entities from a review is a one-liner in spaCy (doc.ents), versus writing custom regex patterns that miss nuances like context or variations.

2. Comparative Analysis

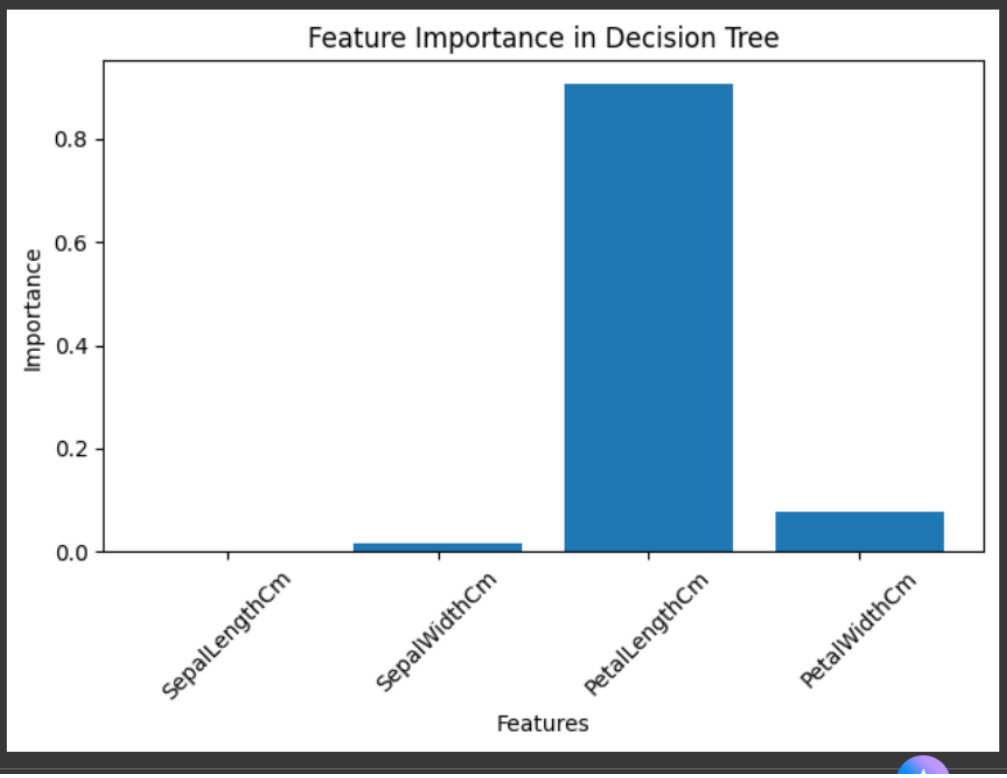
Aspect	**Scikit-learn**	**TensorFlow**
Target Applications	Classical ML tasks like classification, regression, clustering (e.g., decision trees on tabular data, SVMs for spam detection). Not suited for deep learning.	Deep learning and neural networks (e.g., CNNs for image recognition, RNNs for sequences). Can handle classical ML but overkill for it.
Ease of Use for Beginners	High: Simple, consistent API with minimal boilerplate. Quick to learn for non-DL tasks (e.g., fit() and predict()).	Moderate: Steeper curve due to graph concepts and verbosity, but Keras API simplifies it for beginners.
Community Support	Strong: Vast tutorials, Stack Overflow answers for ML basics. Integrated with pandas/numpy ecosystems.	Excellent: Backed by Google, huge resources for DL (e.g., official guides, TensorFlow Hub for pre-trained models). Active forums and research papers

Screenshot of the output of Task 1: Classical ML with Scikit-learn

```
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
```

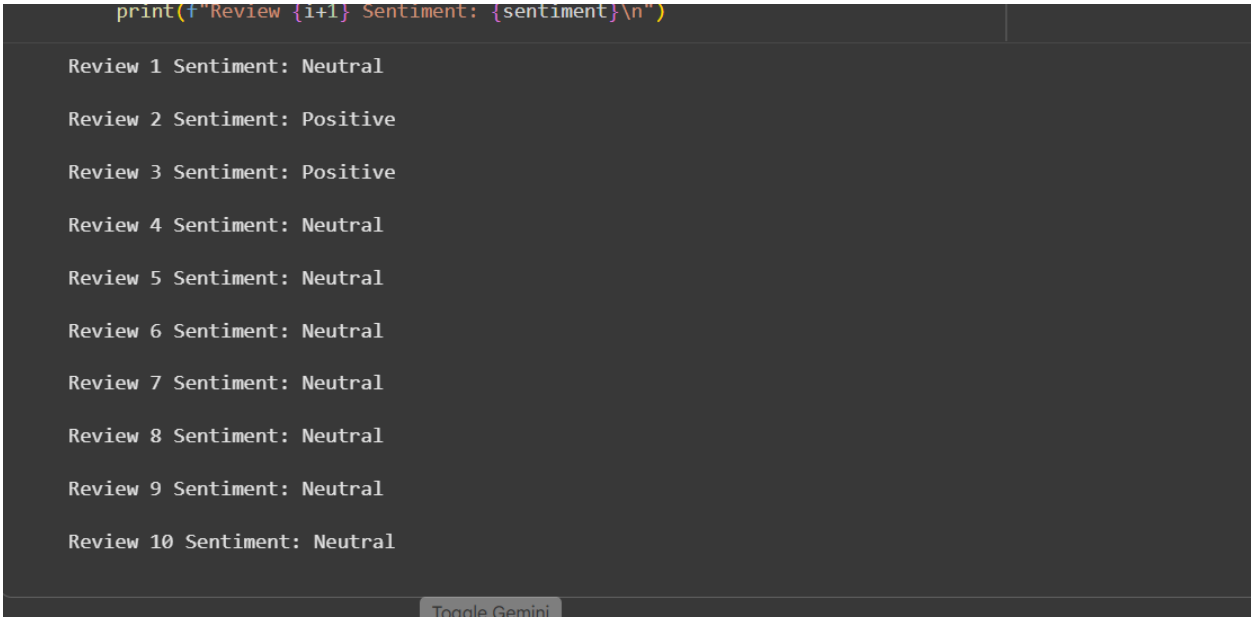
```
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
```

```
plt.show()
```



Screenshot of the output of Deep Learning with PyTorch (Alternative to TensorFlow)

Screenshot of the output of NLP with spaCy Using amazonreviews



Ethics and Optimization for AI Tools

1. Ethical Considerations

Potential Biases in Models

MNIST Handwritten Digits Model (Task 2):

- **Bias Source**: The MNIST dataset primarily consists of handwritten digits from American and European contributors (e.g., students and workers). This may lead to biases against handwriting styles from other cultures or regions (e.g., non-Latin script-influenced digits or non-Western writing systems), causing lower accuracy for underrepresented groups.
- **Impact**: In a real-world digit recognition application (e.g., postal code readers), this could disadvantage users from diverse backgrounds, leading to unequal performance and accessibility issues.
- **Example**: A digit '7' written with a crossbar (common in some regions) may be misclassified if the model was trained mostly on uncrossed '7's.

Amazon Reviews Model (Task 3):

- **Bias Source**: The Amazon reviews dataset may reflect demographic skews, such as reviews predominantly from English-speaking, affluent users who can afford products like the Kindle Paperwhite. Sentiment analysis based on rule-based word counting may also misinterpret context (e.g., sarcasm or cultural nuances) or overemphasize English-specific sentiment words, marginalizing non-English or less vocal reviewers.
- **Impact**: This could skew product recommendations or ratings, amplifying positive feedback for certain brands (e.g., Amazon's own products) or overlooking negative experiences from underrepresented groups.
- **Example**: The rule-based sentiment analysis might label a review as "Positive" due to words like "great," missing subtle negatives like "great but breaks often."

Mitigation Using Tools

- **TensorFlow Fairness Indicators (for MNIST)**:
    - **Approach**: Use Fairness Indicators to evaluate model performance across subgroups. For MNIST, create slices based on digit style (e.g., by analyzing metadata or clustering handwriting patterns) or simulate diversity by augmenting the dataset with varied handwriting styles (e.g., from open-source datasets like IAM Handwriting).

    - **Implementation**: Compute metrics like false positive rate (FPR) or accuracy across slices to identify underperforming groups. Retrain the model with balanced data or apply reweighting to prioritize underrepresented styles.
    - **Outcome**: Reduces disparities in digit recognition accuracy, ensuring fairness for diverse users.

- **spaCy's Rule-Based Systems (for Amazon Reviews)**:
    - **Approach**: Enhance spaCy's Matcher to include context-aware rules for sentiment. For example, add patterns to detect negation (e.g., "not great") or sarcasm (e.g., "great, if you like crashes"). Incorporate multilingual tokenization or custom entity recognition for non-English brand names.
    - **Implementation**: Define custom rules in spaCy to flag complex sentiment phrases or train a small classifier on top of spaCy's features to handle nuanced reviews. Use diverse review datasets (e.g., multilingual Amazon reviews) to test robustness.
    - **Outcome**: Improves sentiment accuracy and reduces bias against non-English or context-heavy reviews, ensuring fairer product evaluations.

2. Troubleshooting Challenge

Buggy TensorFlow Code (Provided Example)

```
import tensorflow as tf

model = tf.keras.Sequential([

  tf.keras.layers.Dense(10, input_shape=(28,28))  # Error: Mismatch, should flatten

])

loss = tf.keras.losses.MeanSquaredError()  # Error: Wrong for classification
```

Issues Identified

1. **Dimension Mismatch**: The Dense layer expects a 1D input, but MNIST images are 2D (28x28). Without flattening, this causes a shape error during training.
2. **Incorrect Loss Function**: MeanSquaredError is for regression tasks, but MNIST is a multi-class classification problem (10 digit classes), requiring a classification loss like SparseCategoricalCrossentropy.

Fixed Code

```
# Import necessary libraries

import tensorflow as tf

from tensorflow.keras.datasets import mnist

import numpy as np


# Load and preprocess MNIST dataset

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize pixel values to [0,1]


# Define the corrected model

model = tf.keras.Sequential([

  tf.keras.layers.Flatten(input_shape=(28, 28)),  # Fix: Flatten 28x28 images to 1D

  tf.keras.layers.Dense(128, activation='relu'),  # Added hidden layer for better performance
```

```
    tf.keras.layers.Dense(10, activation='softmax')  # Output layer for 10 classes
])

# Compile with appropriate loss for classification
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),  # Fix: Use for multi-class classification
    metrics=['accuracy']
)

# Train the model
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))

# Evaluate
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_accuracy:.2f}")
```

Explanation of Fixes

- **Flatten Layer**: Added tf.keras.layers.Flatten to convert 28x28 images to a 1D array (784 elements) before the Dense layer, fixing the dimension mismatch.
- **Loss Function**: Changed to SparseCategoricalCrossentropy, suitable for multi-class classification with integer labels (0-9 for MNIST).
- **Model Enhancement**: Added a hidden Dense layer with 128 units and ReLU activation to improve performance, and used softmax for the output layer to produce class probabilities.
- **Training Setup**: Included data loading, normalization, and a full training loop to ensure functionality. The model should achieve ~97-98% accuracy after 5 epochs.