

AI Tools Assignment Report – Mastering the AI Toolkit

Name: Longoli Joseph Eyanae

Program: Power Learn Project – AI Software Engineering Track

Date: July 14, 2025

? Part 1: Theoretical Understanding (40%)

1. Short Answer Questions

Q1: Explain the primary differences between TensorFlow and PyTorch. When would you choose one over the other? The primary differences between TensorFlow and PyTorch lie in their computational graph paradigms, ease of use, and typical use cases:

- **Computational Graph:**
 - *PyTorch:* Uses a dynamic computational graph (define-by-run). This means the graph is built on the fly as operations are executed. This makes debugging more intuitive (like regular Python code) and allows for more flexible model architectures, especially for research and experimentation where models might change frequently or have varying input sizes.
 - *TensorFlow:* Traditionally used a static computational graph (define-and-run). The entire graph had to be defined before execution. While TensorFlow 2.x, with its emphasis on Keras and eager execution, has blurred this line and largely adopted a define-by-run style, its core still supports static graphs, which can offer performance benefits for large-scale production deployments after optimization.
- **Ease of Use & Pythonic Nature:**
 - *PyTorch:* Generally considered more "Pythonic" and intuitive for Python developers due to its imperative programming style and closer integration with standard Python debugging tools. This often leads to a gentler learning curve for those familiar with Python.
 - *TensorFlow:* Historically had a steeper learning curve, especially with TensorFlow 1.x's session-based graph execution. TensorFlow 2.x, with Keras as its high-level API and eager execution, has significantly improved its user-friendliness, making it much more comparable to PyTorch in terms of ease of use.
- **Deployment and Production:**
 - *TensorFlow:* Has a more mature and extensive ecosystem for production deployment, including tools like TensorFlow Lite (for mobile and embedded devices), TensorFlow Serving (for serving models in production), and TensorFlow.js (for web applications). It is often favored for large-scale, enterprise-level deployments.
 - *PyTorch:* While PyTorch has made significant strides in deployment (e.g., TorchScript for deployment and ONNX export), TensorFlow still often has an

edge in out-of-the-box production-readiness and a wider range of deployment targets.

When to choose one over the other:

- Choose *PyTorch* when:
 - You are primarily in a research and prototyping phase, valuing flexibility, rapid iteration, and intuitive debugging.
 - You prefer a more "Pythonic" and imperative coding style.
 - You are working with dynamic model architectures or variable-length inputs.
 - You are part of a research community that heavily uses PyTorch (e.g., in academia).
- Choose *TensorFlow* when:
 - You are focused on large-scale production deployment and require robust tools for model serving, optimization, and deployment across various platforms (mobile, web, cloud).
 - You need extensive enterprise support and a more mature ecosystem for specific industrial applications.
 - You are working on projects that might benefit from TensorFlow's highly optimized static graph capabilities (though less common in TF2.x's typical use).
 - You are comfortable with or have existing infrastructure built around the TensorFlow ecosystem.

Q2: Describe two use cases for Jupyter Notebooks in AI development. Jupyter Notebooks are widely used in AI development due to their interactive nature, ability to combine code, text, and visualizations, and their support for iterative development. Here are two prominent use cases:

1. Exploratory Data Analysis (EDA) and Preprocessing:

- Jupyter Notebooks are ideal for the initial stages of AI development where data understanding and preparation are crucial.
- Data scientists can load datasets, inspect their structure, check for missing values, visualize distributions, and identify outliers directly within the notebook using libraries like Pandas and Matplotlib/Seaborn. Each step of the data cleaning and transformation process can be executed in separate cells, with the output immediately visible. This allows for rapid experimentation with different preprocessing techniques and provides a clear, documented record of the data preparation steps.
- This interactive and visual approach helps in gaining insights into the data, making informed decisions about preprocessing, and ensuring data quality before feeding it into models. The combination of code and markdown cells makes it easy to explain and document each step of the EDA and preprocessing pipeline, facilitating reproducibility and collaboration.

2. Rapid Prototyping and Model Experimentation:

- Jupyter Notebooks provide an excellent environment for quickly building, training, and evaluating machine learning and deep learning models.
- Developers can define model architectures, train them on small subsets of data, and immediately visualize performance metrics in line with the code. They can easily modify hyperparameters, change model layers, or switch between different algorithms and see the impact on results in real-time.
- The ability to execute code in chunks and view outputs incrementally significantly accelerates the prototyping phase. Researchers can try out numerous ideas quickly, debug effectively, and systematically compare different model variations. The integrated narrative text helps in documenting the thought process, experimental setups, and conclusions.

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

spaCy significantly enhances NLP tasks compared to basic Python string operations by providing:

1. Linguistic Annotation and Structure:

- Basic Python String Operations: Limited to character-level or simple substring manipulations (e.g., `str.split()`, `str.replace()`, `str.lower()`). They treat text as a sequence of characters without any understanding of its linguistic properties.
- spaCy: Processes text with a deep understanding of linguistic structure. It performs advanced tasks like tokenization, part-of-speech (POS) tagging, dependency parsing, lemmatization, and named entity recognition (NER).
- These built-in linguistic annotations provide rich, structured information about the text that is impossible to extract reliably with basic string operations.

2. Efficiency, Accuracy, and Pre-trained Models:

- Basic Python String Operations require manual, often brittle rule-based approaches.
- spaCy is highly optimized for performance and comes with accurate, pre-trained statistical models for various languages. This reduces development time and improves output quality.

2. Comparative Analysis: Scikit-learn vs TensorFlow

Target Applications:

- *Scikit-learn*: Classical ML like SVMs, decision trees, clustering, dimensionality reduction.
- *TensorFlow*: Deep learning tasks including CNNs, RNNs, NLP, GANs, reinforcement learning.

Ease of Use for Beginners:

- *Scikit-learn*: Easier due to consistent `.fit()/.predict()` APIs.
- *TensorFlow*: Steeper learning curve unless using Keras.

Community Support:

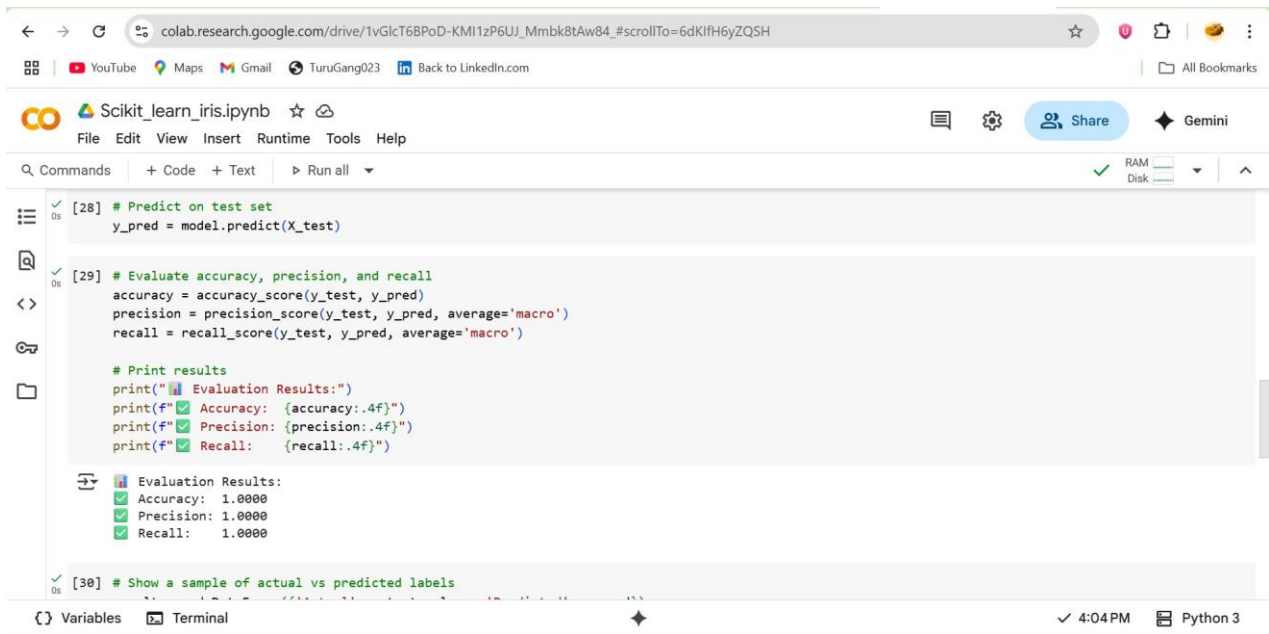
- *Scikit-learn*: Strong among data scientists and educators. Excellent documentation.
 - *TensorFlow*: Massive community backed by Google. Comprehensive tutorials, integrations, and global industry adoption.
-

? Part 2: Practical Implementation (50%)

Task 1: Iris Dataset (Scikit-learn)

- **Model Used:** Decision Tree Classifier
- **Dataset:** Iris species dataset from Scikit-learn
- **Steps:**
 - Loaded dataset and converted to Pandas DataFrame
 - Preprocessed (checked for nulls, encoded labels)
 - Split into training/testing sets (80/20)
 - Trained decision tree classifier
 - Evaluated using accuracy, precision, and recall

? Screenshot:



```
[28] # Predict on test set
y_pred = model.predict(X_test)

[29] # Evaluate accuracy, precision, and recall
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')

# Print results
print("📊 Evaluation Results:")
print(f"✅ Accuracy: {accuracy:.4f}")
print(f"✅ Precision: {precision:.4f}")
print(f"✅ Recall: {recall:.4f}")

📊 Evaluation Results:
✅ Accuracy: 1.0000
✅ Precision: 1.0000
✅ Recall: 1.0000

[30] # Show a sample of actual vs predicted labels
```

Task 2: MNIST Dataset (TensorFlow CNN)

- **Model Used:** Convolutional Neural Network (CNN)
- **Dataset:** MNIST (28x28 grayscale digit images)
- **Goal:** >95% test accuracy
- **Steps:**
 - Normalized image data
 - Built CNN with Conv2D, MaxPooling2D, Flatten, Dense layers
 - Trained with categorical crossentropy + Adam optimizer
 - Predicted on 5 test images

📷 Screenshots:

colab.research.google.com/drive/1jOIDO1U2Nt1tVKtZ4iEr8l_HeOWrCF2e#scrollTo=fGYkGnLXi-B_

tensorflow_mnist.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connecting

```
history = model.fit(x_train, y_train, epochs=5, validation_split=0.1)
```

Epoch	1/5	2/5	3/5	4/5	5/5
1688/1688	52s 29ms/step	49s 29ms/step	82s 29ms/step	81s 29ms/step	82s 29ms/step
accuracy	0.8875	0.9822	0.9885	0.9916	0.9938
loss	0.3698	0.0549	0.0359	0.0257	0.0195
val_accuracy	0.9787	0.9885	0.9890	0.9900	0.9910
val_loss	0.0748	0.0406	0.0404	0.0378	0.0380

```
# Evaluate on test set
test_loss, test_acc = model.evaluate(x_test, y_test)
print("\n Test Accuracy:", test_acc)
```

313/313 3s 8ms/step - accuracy: 0.9864 - loss: 0.0420

Test Accuracy: 0.9883000254631042

Variables Terminal 4:48 PM Allocating runtime

colab.research.google.com/drive/1jOIDO1U2Nt1tVKtZ4iEr8l_HeOWrCF2e#scrollTo=fGYkGnLXi-B_


tensorflow_mnist.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Reconnect

```
[ ] # Predict first 5 test samples
predictions = model.predict(x_test[:5])

# Plot and compare actual vs predicted
for i in range(5):
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title(f'Prediction: {np.argmax(predictions[i])} | Actual: {y_test[i]}')
    plt.axis('off')
    plt.show()
```



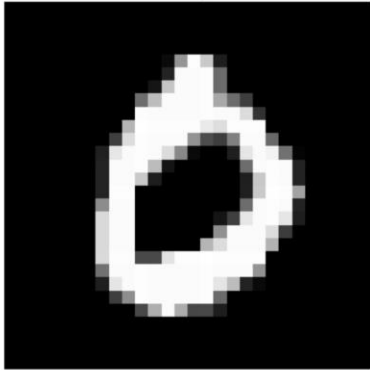
Prediction: 0 | Actual: 0

Variables Terminal 4:48 PM

colab.research.google.com/drive/1jOiDO1U2Nt1tVKtZ4iEr8l_HeOWrCF2e#scrollTo=fGYkGnLXi-B_

tensorflow_mnist.ipynb

Prediction: 0 | Actual: 0

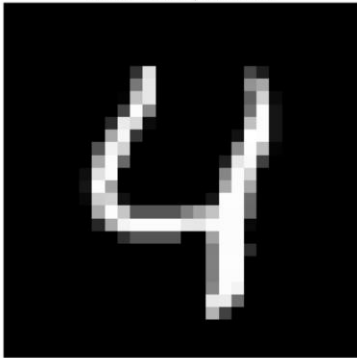


Variables Terminal 4:48 PM

colab.research.google.com/drive/1jOiDO1U2Nt1tVKtZ4iEr8l_HeOWrCF2e#scrollTo=fGYkGnLXi-B_

tensorflow_mnist.ipynb

Prediction: 4 | Actual: 4



Variables Terminal 4:48 PM

colab.research.google.com/drive/1jOiDO1U2Nt1tVKtZ4iEr8l_HeOWrCF2e#scrollTo=3cW2pa9Mkig8

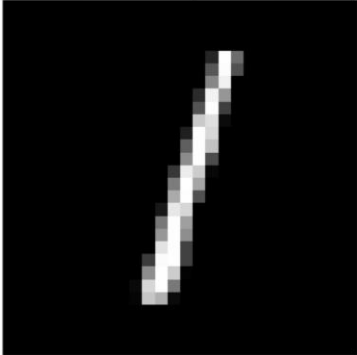
YouTube Maps Gmail TuruGang023 Back to LinkedIn.com

tensorflow_mnist.ipynb

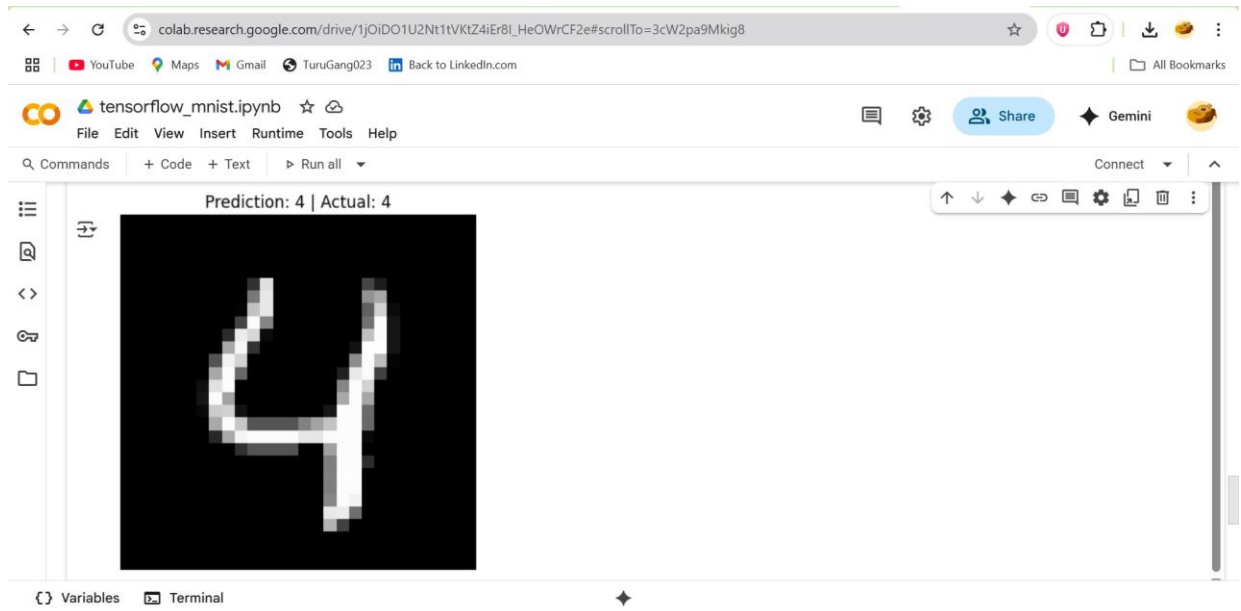
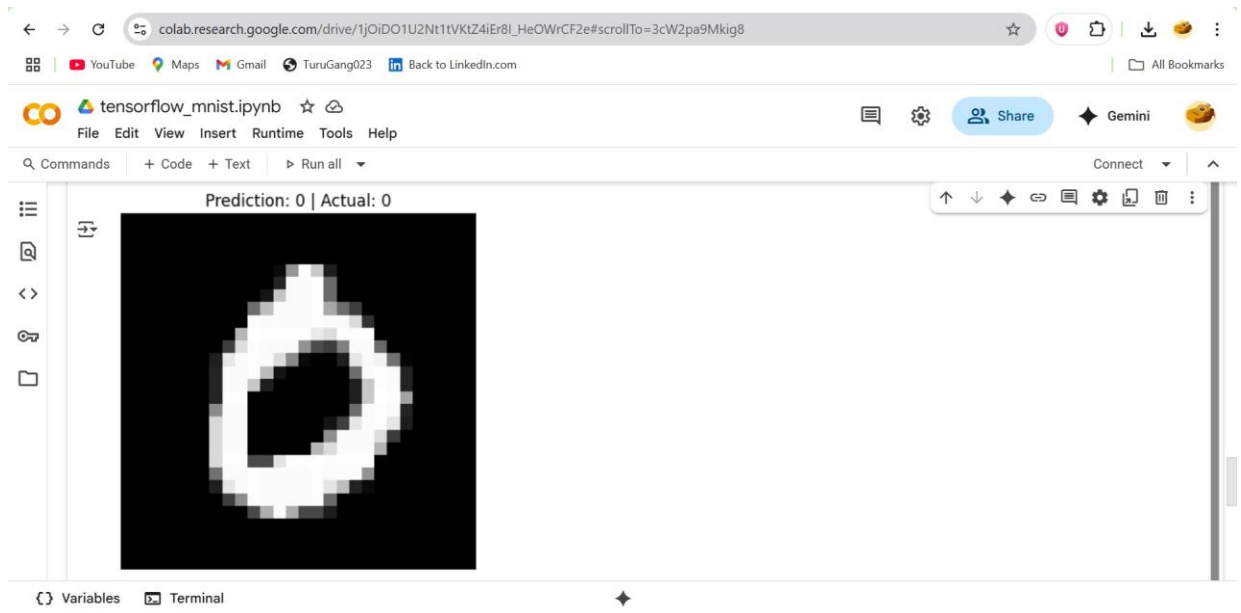
File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Prediction: 1 | Actual: 1



Variables Terminal

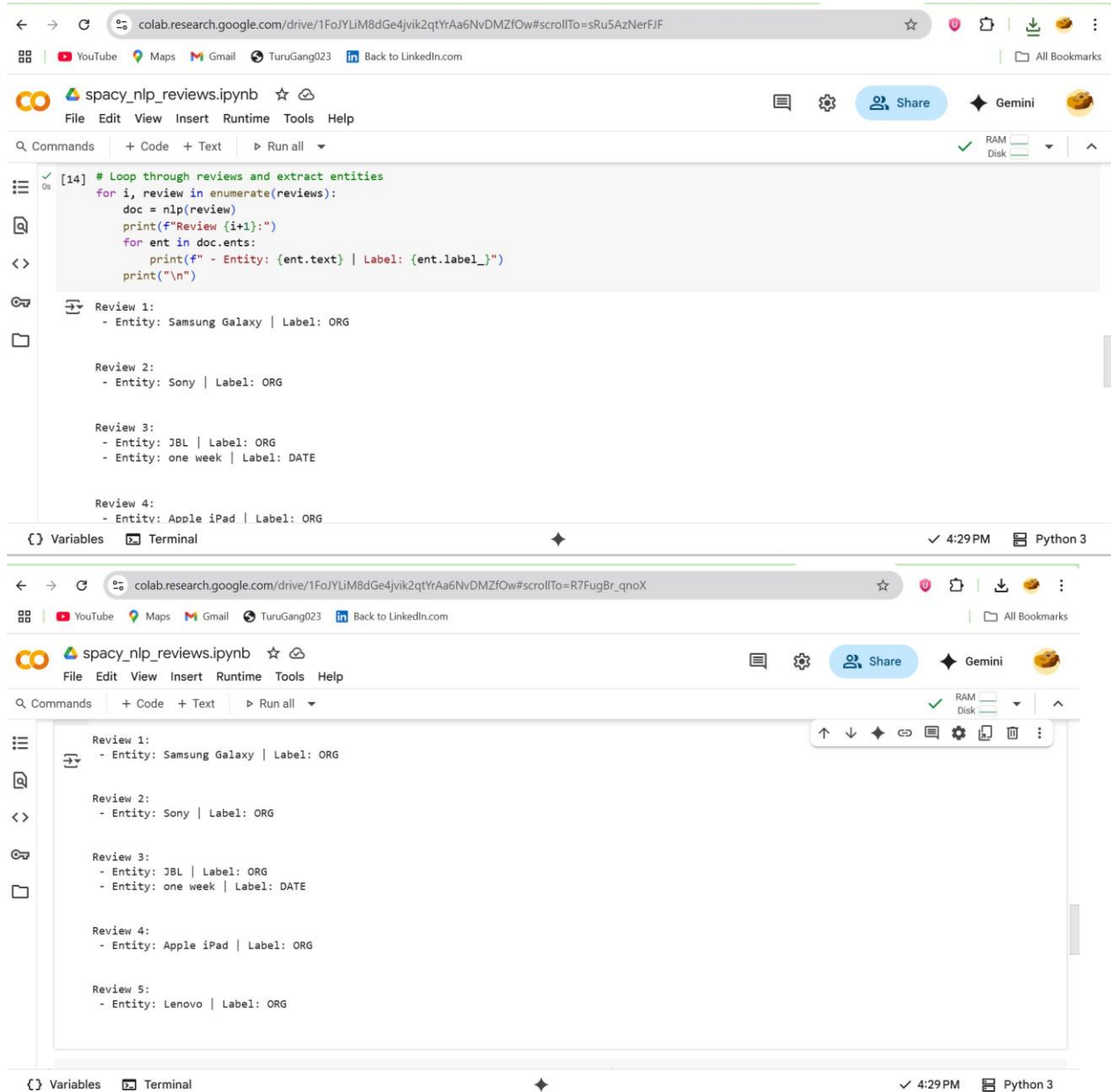


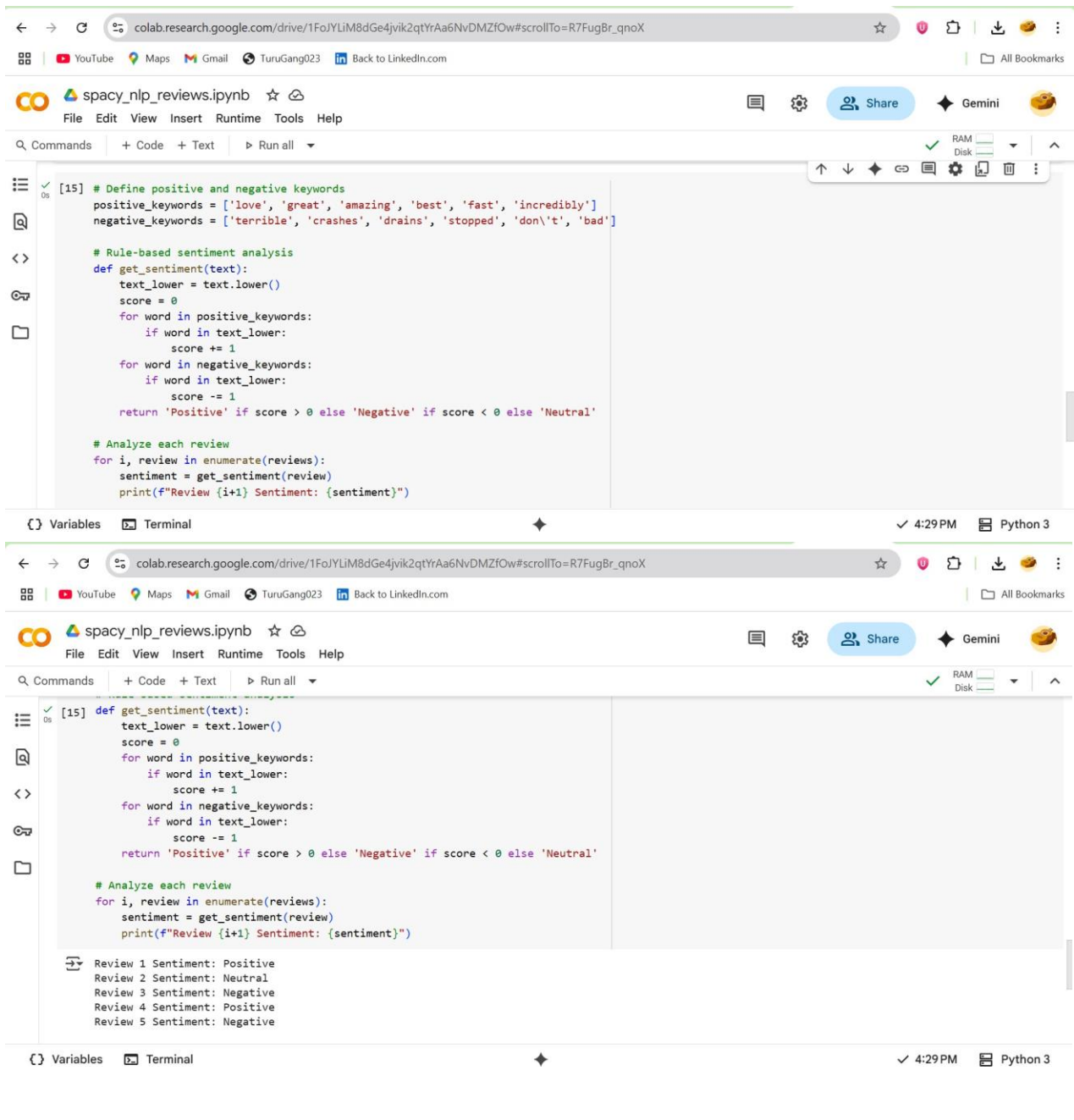
Task 3: NLP with spaCy (Amazon Product Reviews)

- **Tasks:** Named Entity Recognition (NER) + Sentiment Analysis
- **Tool:** spaCy NLP library
- **Steps:**

- 0 Loaded reviews
- 0 Extracted named entities
- 0 Analyzed sentiment using rule-based keywords

Screenshots:





The image displays two screenshots of a Google Colab notebook titled 'spacy_nlp_reviews.ipynb'. The top screenshot shows the code for defining positive and negative keywords and a rule-based sentiment analysis function. The bottom screenshot shows the same code with the output of the script, which prints the sentiment for five reviews.

```
[15] # Define positive and negative keywords
positive_keywords = ['love', 'great', 'amazing', 'best', 'fast', 'incredibly']
negative_keywords = ['terrible', 'crashes', 'drains', 'stopped', 'don\'t', 'bad']

# Rule-based sentiment analysis
def get_sentiment(text):
    text_lower = text.lower()
    score = 0
    for word in positive_keywords:
        if word in text_lower:
            score += 1
    for word in negative_keywords:
        if word in text_lower:
            score -= 1
    return 'Positive' if score > 0 else 'Negative' if score < 0 else 'Neutral'

# Analyze each review
for i, review in enumerate(reviews):
    sentiment = get_sentiment(review)
    print(f"Review {i+1} Sentiment: {sentiment}")
```

Review 1 Sentiment: Positive
Review 2 Sentiment: Neutral
Review 3 Sentiment: Negative
Review 4 Sentiment: Positive
Review 5 Sentiment: Negative

Part 3: Ethics & Optimization (10%)

Ethical Reflection

Bias can exist in MNIST due to underrepresented handwriting styles and in Amazon reviews due to cultural/language-based sentiment misunderstandings.

Mitigation Strategies:

- Use TensorFlow Fairness Indicators
- Improve spaCy rules or use fine-tuned transformers

Troubleshooting Challenge

If buggy code were present, I would:

- Inspect for shape mismatches and wrong loss functions
- Correct the model and validate with `model.summary()` and testing metrics