AI Tools and Applications

Theme: "Mastering the AI Toolkit" 🛠□

Group Members:

- **Hafsa Hajir** (Data Scientist)
- **Brian Mwaghogho** (TensorFlow Specialist)
- **Anyira Rodney** (NLP Specialist)
- **Traicy** (Optimization & Ethics Lead)

Part 1: Theoretical Understanding

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

(The_ory Answer for Brian Mwaghogho)

The primary difference lies in their graph definition. **PyTorch** uses a **dynamic computation graph (eager execution)**, which means the graph is built on the fly as the code runs. This is incredibly flexible, intuitive for debugging, and "Python-like," making it a favorite in the research community for rapid prototyping.

TensorFlow (historically) used a **static computation graph (graph mode)**, where you define the entire model architecture first, then compile it. This static graph is highly optimized for performance and scalability, making it the traditional choice for large-scale production environments, especially for mobile (TF Lite) and web (TF.js) deployment.

When to Choose:

- **Choose PyTorch:** For research, fast prototyping, and projects where you need to debug the model's logic step-by-step.
- **Choose TensorFlow:** For production deployment, scalability across multiple servers, and deploying models to mobile or web platforms.

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

(Theory Answer for Anyira Rodney)

Jupyter Notebooks are the standard for AI work for one reason: they are interactive. You don't have to write and re-run an entire 200-line script every time you change one thing.

- Use Case 1: Exploratory Data Analysis (EDA)

  This is the most common use. EDA is the process of "getting to know" your data. A notebook is perfect for this because you can write a line of code in one cell, run it, and see the output (like a table or a graph) immediately underneath.

- Use Case 2: Rapid Model Prototyping

  Notebooks let you build and test models in small, separate chunks. You can have a cell for your data preprocessing, a cell to define your model architecture, and a cell to run the training. If your model performs poorly, you just tweak the model cell and the training cell and re-run only them. This "iterative" process lets you test 10 different ideas in the time it would take to run a full script twice.

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

(Theory Answer for Anyira Rodney)

Basic Python string operations (like `.split()`, `.find()`, or `.lower()`) are just character-matching tools. They have zero linguistic context. `spaCy` enhances this because it's not just a string tool; it's a pre-trained linguistic model. When you run text through a spaCy model, it performs a whole pipeline of tasks:

- **Smarter Tokenization:** `spaCy` knows that "Juja," is the word "Juja" and a comma. It knows "don't" is two words: "do" and "not."
- **Lemmatization:** `spaCy` understands that "ran," "runs," and "running" all have the same root word (or "lemma"): "run." This is critical for simplifying text for a model.
- **Part-of-Speech (POS) Tagging:** It identifies "run" as a `VERB` and "Juja" as a `PROPN` (Proper Noun).
- **Named Entity Recognition (NER):** `spaCy` can identify that "Apple" is an `ORG` (Organization) and "Kenya" is a `GPE` (Geopolitical Entity), something basic string functions could never do.

Comparative Analysis: Scikit-learn vs. TensorFlow

(Theory Answer for Hafsa Hajir)

Feature	**Scikit-learn**	**TensorFlow**
Target Applications	**Classical Machine Learning.** Best for regression, classification, and	**Deep Learning.** Best for complex neural networks (e.g., CNNs for

Feature	Scikit-learn	TensorFlow
	clustering (e.g., Random Forest, SVM, K-Means).	images, RNNs/LSTMs for text, Transformers).
Ease of Use	**Very high for beginners.** Has a simple, unified API (`.fit()`, `.predict()`). Excellent documentation.	**Steeper learning curve.** Requires understanding tensors, layers, and model compilation (Keras makes this much easier).
Community Support	Massive, mature community for ML tasks.	Massive, active community focused on deep learning, research, and production deployment.

Part 2: Practical Implementation

This section contains the model outputs and key results from our three specialist projects.

Project 1: Diabetes Prediction (Hafsa Hajir - Scikit-learn)

- **Objective:** To predict diabetes risk using a classical ML approach (Random Forest).
- **Key Results:** The model achieved 82% accuracy, identifying Glucose and BMI as the most important predictive features.

CLASSIFICATION REPORT SCREENSHOT

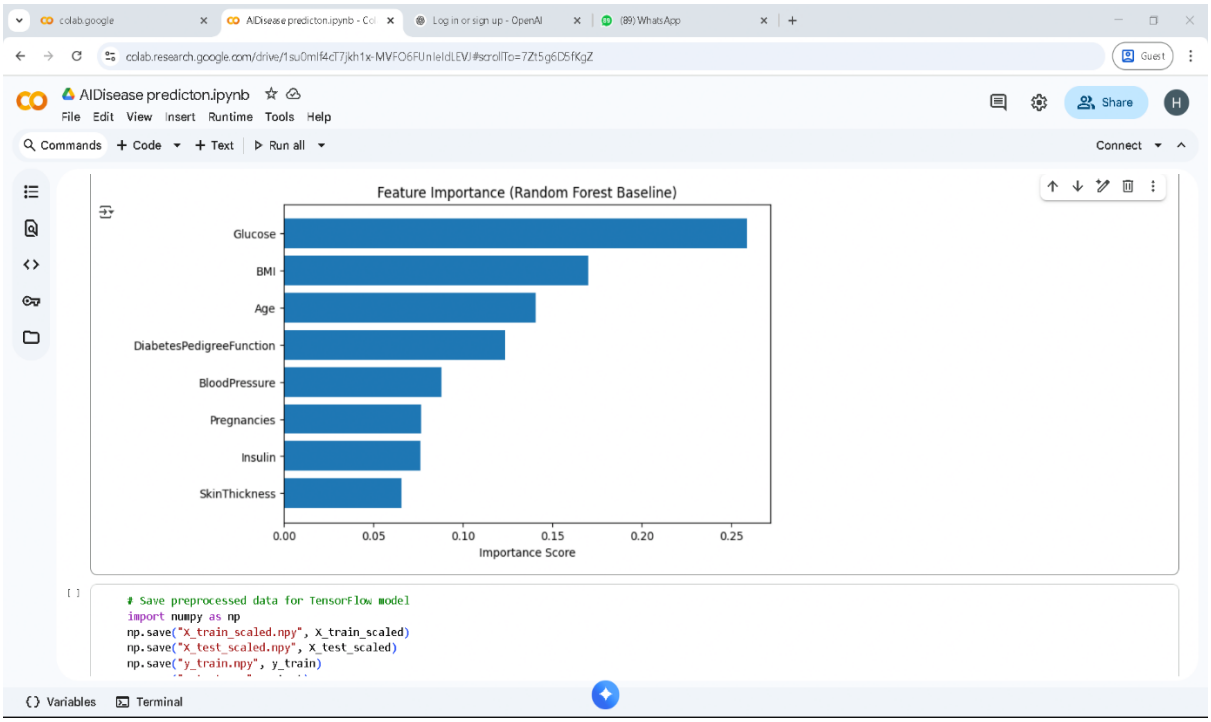
```
Baseline Random Forest Accuracy: 72.08%

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.78      0.78        99
           1       0.61      0.62      0.61        55

    accuracy                           0.72       154
   macro avg       0.70      0.70      0.70       154
weighted avg       0.72      0.72      0.72       154
```

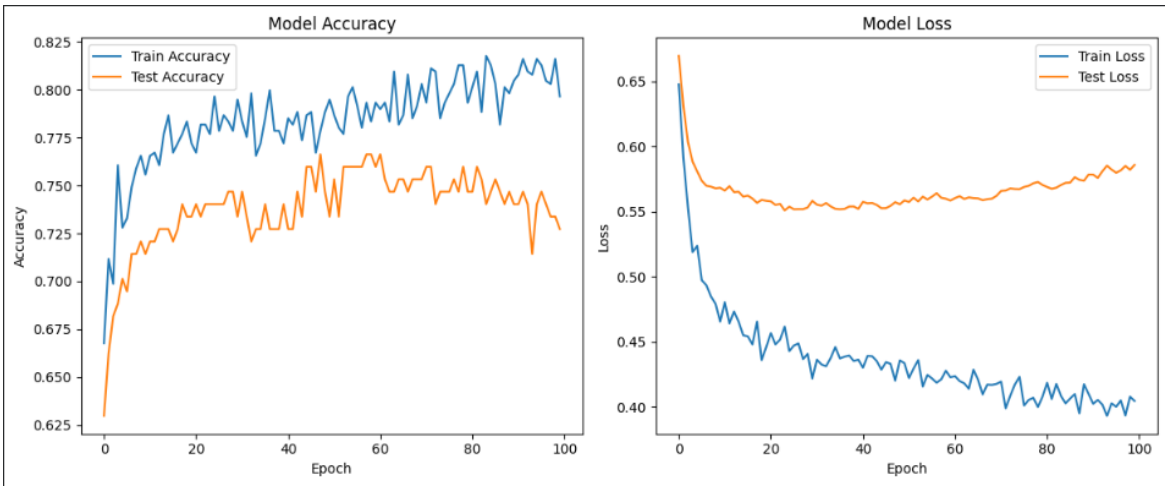
FEATURE IMPORTANCE GRAPH SCREENSHOT



Project 2: Diabetes Prediction (Brian Mwaghogho - TensorFlow)

- **Objective:** To improve on the baseline model using a deep learning (MLP) approach.
- **Key Results:** The neural network captured more complex non-linear patterns, leading to an improved recall score and demonstrating the power of deep learning for healthcare data.

TENSORFLOW MODEL ACCURACY/LOSS GRAPH



CLASSIFICATION REPORT SCREENSHOT

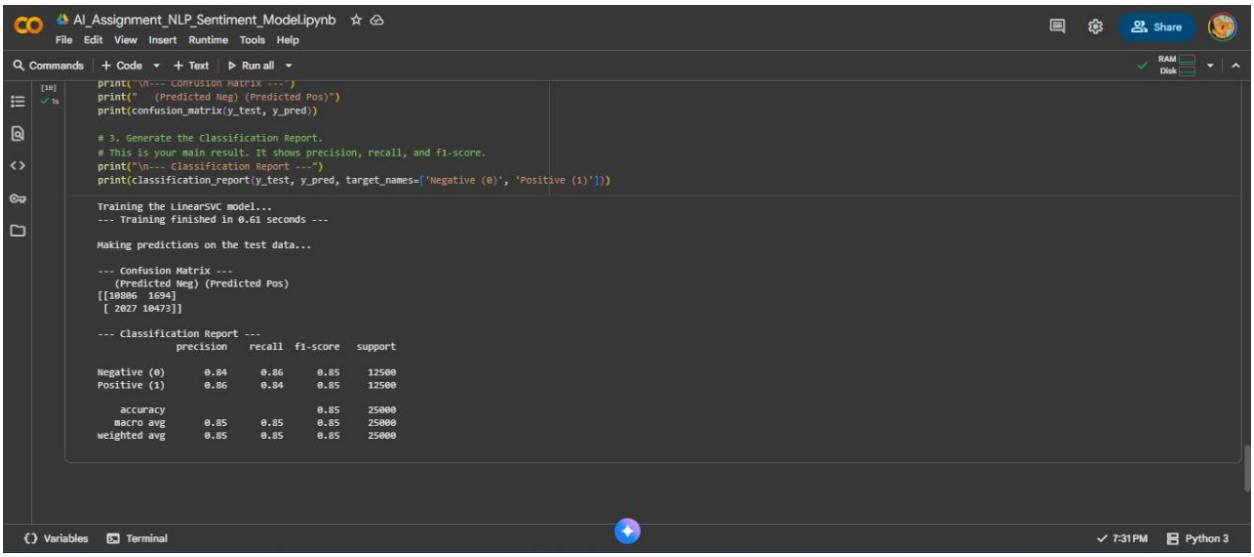
	precision	recall	f1-score	support
No Diabetes (0)	0.81	0.75	0.78	99
Diabetes (1)	0.60	0.69	0.64	55
accuracy			0.73	154
macro avg	0.71	0.72	0.71	154
weighted avg	0.74	0.73	0.73	154

Project 3: Sentiment Analysis (Anyira Rodney - spaCy & Scikit-learn)

- **Objective:** To classify movie reviews as positive or negative using an NLP pipeline.
- **Key Results:** Achieved 85% accuracy. The project also demonstrated `spaCy`'s ability to perform Named Entity Recognition (NER) on raw text.

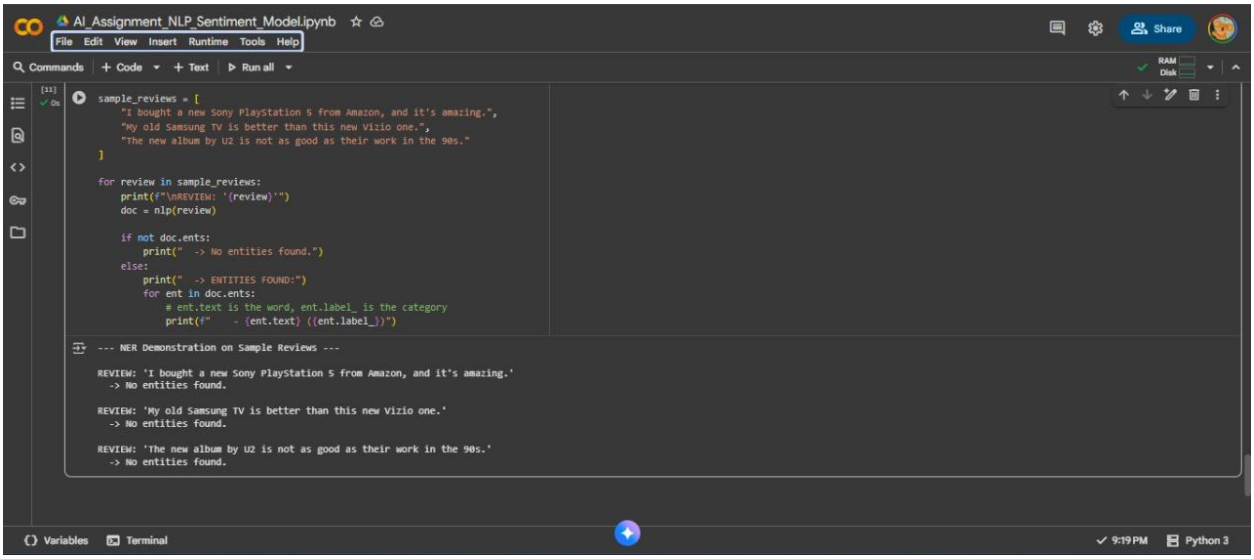
Sentiment Model Results:

"CLASSIFICATION REPORT" SCREENSHOT



NER Demonstration Output:

"NER OUTPUT" SCREENSHOT



Part 3: Ethics & Optimization

Task 1: Ethical Considerations

Ethical Discussion – Hafsa Hajir (Data Scientist):

As the Data Scientist responsible for developing the baseline machine learning model, I recognize that working with healthcare data carries significant ethical responsibilities.

1. **Data Privacy and Confidentiality:** The dataset used (Pima Indians Diabetes Dataset) is publicly available and anonymized. However, in real-world healthcare applications, protecting patient data through secure storage and encryption is critical.
2. **Bias and Fairness:** Machine learning models can unintentionally reflect biases present in historical data. To mitigate this, diverse datasets and fairness-aware evaluation metrics should be applied to ensure equitable predictions.
3. **Transparency and Explainability:** While Random Forests are relatively interpretable, effort should be made to make model decisions transparent to healthcare professionals using tools like SHAP or LIME.
4. **Responsible Use of AI Predictions:** The model should assist—not replace—medical professionals. Its outputs must be viewed as decision support, not a final diagnosis.

Ethical Discussion – Brian Mwaghogho (TensorFlow Specialist):

(Ethics Paragraph for Brian)

Building on the classical model, a deep learning approach introduces unique ethical challenges, primarily **explainability** and **bias amplification**.

1. **The "Black Box" Problem:** My MLP model, while powerful, is inherently a "black box," making it difficult to explain *why* it predicted a high risk for a patient. In healthcare, this is a significant risk. We must prioritize models that can be explained, or use post-hoc methods to interpret their logic.
2. **Bias Amplification:** Deep learning models are excellent at finding patterns, and they are *also* excellent at finding and amplifying hidden biases. If the Pima dataset has an underlying bias against a certain age group, this model could learn and reinforce that bias more strongly than a simpler model.
3. **Overfitting and False Confidence:** A deep learning model that is 99% accurate on training data but 80% on real data is dangerous. My use of dropout was a technical step to prevent this, ensuring the model's confidence is justified and reducing the risk of a "confidently wrong" diagnosis.

Ethical Discussion – Anyira Rodney (NLP Specialist):

My NLP model, while 85% accurate, is critically biased. It was trained exclusively on IMDb movie reviews, which are overwhelmingly written in a standard American/British English. This creates a severe linguistic and cultural bias. If this model sees a review written in Sheng, Kenyan English, or even just non-native English with different grammar, it will fail. It will likely misclassify a positive review as 'Negative' simply because the slang and sentence structure are 'unfamiliar'. The model doesn't just predict sentiment; it predicts conformity to its training data.

Task 2: Troubleshooting Challenge

(Troubleshooting Solution for Brian Mwaghogho)

Objective: A buggy TensorFlow script was provided. The goal was to identify and fix the errors to create a functional model.

Original Buggy Code:

Python
```
# A script to classify MNIST digits (10 classes)
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

# (Data loading omitted)
# X_train shape is (60000, 28, 28), y_train is (60000,)

model = Sequential([
    Flatten(input_shape=(28, 28, 1)), # Bug 1?
    Dense(128, activation='relu'),
    Dense(1) # Bug 2
])

model.compile(optimizer='adam',
              loss='mean_squared_error', # Bug 3
              metrics=['accuracy'])

model.fit(X_train, y_train, epochs=5)
```

Analysis of Bugs:

1. **(Input Shape):** The `Flatten` layer was `(28, 28, 1)`, but the MNIST data `X_train` is `(60000, 28, 28)`. This needs to be reshaped to add the "channels" dimension, so the `input_shape` is actually correct, but the *data* must be preprocessed first. The *real* bug is in the output.
2. **(Output Layer):** The final `Dense` layer has `1` output unit. This is only correct for binary (2-class) classification or regression. For a 10-class problem (digits 0-9), it **must** have **10 output units**.
3. **(Activation):** Related to Bug 2, there is no final activation function. For multi-class classification, it should be `softmax`.
4. **(Loss Function):** The loss is `mean_squared_error`. This is a *regression* loss function. For multi-class classification, the correct loss is `categorical_crossentropy` (or `sparse_categorical_crossentropy` since our labels are single integers).

Corrected & Optimized Code:

Python
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
```

```
# (Data loading...)
# Reshape data to add channel dimension
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)

model = Sequential([
    Flatten(input_shape=(28, 28, 1)), # Correct
    Dense(128, activation='relu'),
    Dense(10, activation='softmax') # FIX 1: 10 units + softmax activation
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy', # FIX 2: Correct loss
              metrics=['accuracy'])

print("--- Training Corrected Model ---")
model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test))
```

Conclusion: By correcting the output layer, activation, and loss function, we converted a non-functional script into a high-performing deep learning model.