1. Short Answer Questions

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

FEATURE	TENSOR FLOW	PYTORCH
Computation Model	Static computation graph (via Graphs)	Dynamic computation graph (eager execution)
Ease of Use	Steeper learning curve, more verbose	Intuitive and Pythonic, easier for newcomers
Deployment	Strong for production (TF Serving, TF Lite)	Some deployment tools, but less mature
Ecosystem	Extensive (e.g., TensorBoard, TFX)	Solid but leaner ecosystem
Community Support	Large, backed by Google	Growing fast, backed by Meta
Performance	Highly optimized, excellent for scale	Great performance, especially in research

Use PyTorch if:

- You're doing rapid prototyping or academic research.
- You want a framework that feels more like native Python and allows more flexibility.
- You prioritize readability and debuggability.

Choose TensorFlow if:

- You're preparing a model for production at scale.
- You need mobile or embedded deployment (e.g., using TensorFlow Lite).
- You want a rich ecosystem with tools for visualization, serving, and optimization.

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

Exploratory Data Analysis (EDA)

Jupyter shines when you're getting familiar with a dataset. You can:

- Load data, visualize distributions, and check for anomalies—all in one place.

- Interleave code with visual output (like charts or tables) and markdown cells for explanations, which makes the analysis easy to share or revisit.

For example, if you're working on a model to predict crop yields, you might use a Jupyter Notebook to inspect weather trends, soil quality metrics, and planting seasons, adjusting and annotating your code as you discover patterns.

Prototyping Machine Learning Models

Building and tweaking models in real time is one of Jupyter's strong suits. It allows you to:

- Rapidly experiment with model architectures and hyperparameters.

- Monitor performance with visual feedback like loss curves or confusion matrices, all within the notebook.

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

🚀 Key Enhancements with spaCy

1. Linguistic Awareness

spaCy processes text using advanced models that understand **grammar, syntax, and semantics**. This includes part-of-speech tagging, lemmatization, and syntactic parsing—features that plain string operations just can’t deliver.

2. Named Entity Recognition (NER)

spaCy can identify real-world entities like names, dates, organizations, and locations out of the box. For example, it can tag “Nairobi” as a `GPE` (geo-political entity), something a regular expression wouldn’t recognize without training.

3. Tokenization with Context

Unlike splitting strings by whitespace, spaCy’s tokenization is context-aware—it handles punctuation, contractions, and complex language constructs much more accurately.

4. Pretrained Pipelines

spaCy comes with optimized, pretrained models for various languages. This means you don’t need to build NLP tools from scratch; you get speed, efficiency, and solid accuracy from the start.

5. Integration with Other Tools

It works beautifully with machine learning libraries (like Scikit-learn) and supports fast annotations, model training, and deployment. It's designed for production use, not just experimentation.

2. Comparative Analysis

- Compare Scikit-learn and TensorFlow in terms of:
    - Target applications (e.g., classical ML vs. deep learning).
    - Ease of use for beginners.
    - Community support.

🎯 Target Applications

Scikit-learn: Best suited for **__classical machine learning__** algorithms such as decision trees, random forests, support vector machines, and logistic regression. It's ideal for structured/tabular data tasks—think customer churn prediction or credit scoring.

TensorFlow: Designed primarily for **__deep learning and neural networks__**, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers. It excels in tasks like image recognition, natural language processing, and time-series forecasting with large-scale data.

👶 Ease of Use for Beginners

Scikit-learn: Extremely beginner-friendly. With its simple and consistent API (`fit()`, `predict()`, `score()`), it's an excellent starting point for those learning ML fundamentals.

TensorFlow: Has a steeper learning curve, especially when working with low-level APIs. However, higher-level interfaces like Keras (which is now integrated into TensorFlow) make it more accessible and user-friendly for newcomers to deep learning.

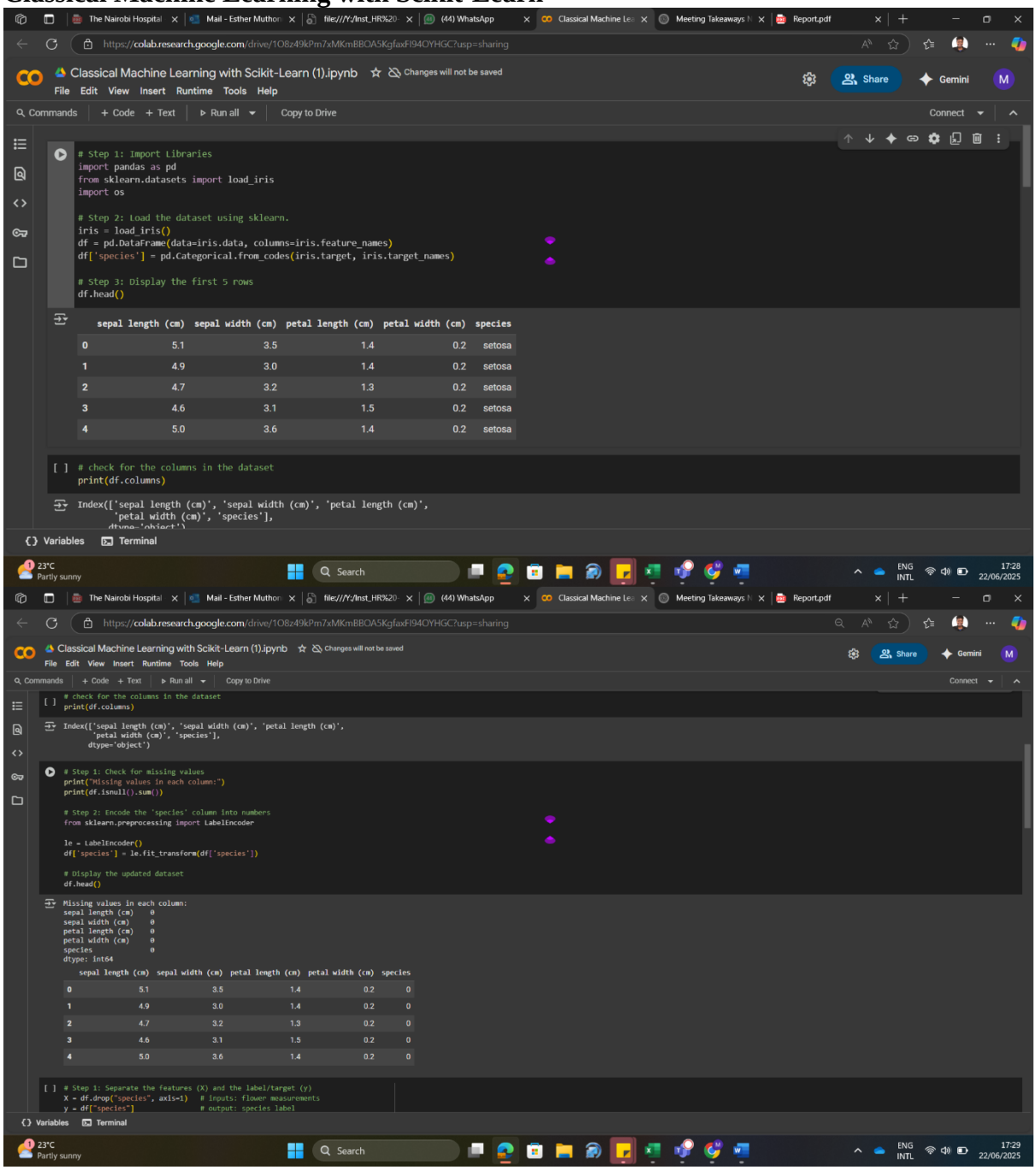
🌍 Community Support

Scikit-learn: Backed by a strong academic and developer community. Since it's been around for a while, there's excellent documentation, numerous tutorials, and a stable release cycle.

TensorFlow: Massive global community, thanks in part to Google's backing. You'll find extensive documentation, active GitHub discussions, and a huge ecosystem of models, datasets, and prebuilt tools.

SCREENSHOTS

Classical Machine Learning with Scikit-Learn



Classical Machine Learning with Scikit-Learn (1).ipynb

```
# Step 1: Import Libraries
import pandas as pd
from sklearn.datasets import load_iris
import os

# Step 2: Load the dataset using sklearn.
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)

# Step 3: Display the first 5 rows
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
# check for the columns in the dataset
print(df.columns)
```

```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)', 'species'],
      dtype='object')
```

Variables Terminal


```
# check for the columns in the dataset
print(df.columns)
```

```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)', 'species'],
      dtype='object')
```

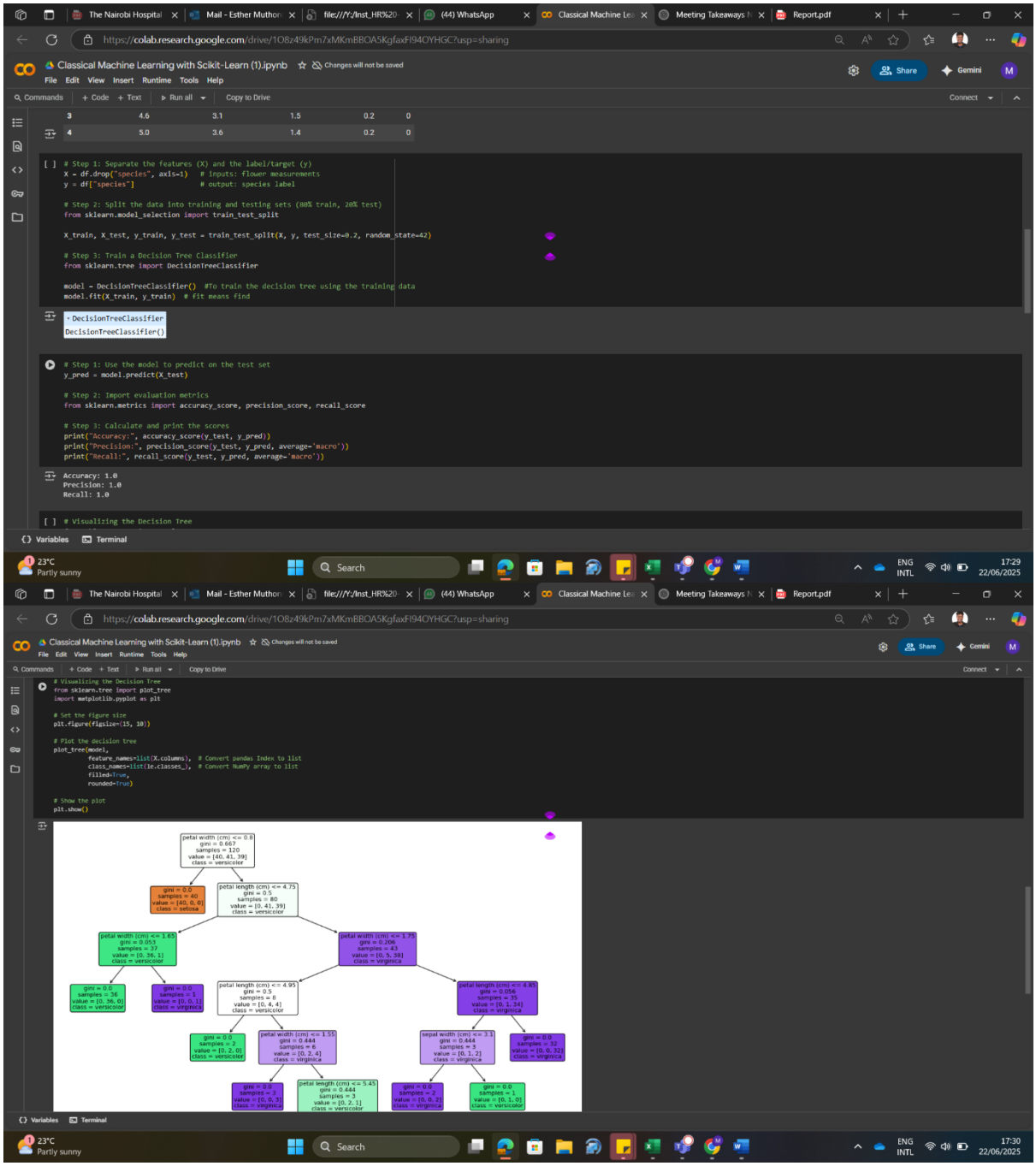
```
# Step 1: Check for missing values
print("Missing values in each column:")
print(df.isnull().sum())

# Step 2: Encode the 'species' column into numbers
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['species'] = le.fit_transform(df['species'])

# Display the updated dataset
df.head()
```

```
Missing values in each column:
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
species              0
dtype: int64
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
# Step 1: Separate the features (X) and the label/target (y)
X = df.drop("species", axis=1)  # Inputs: flower measurements
y = df["species"]               # Output: species label
```

Variables Terminal



. Ethical Considerations

Potential Biases

MNIST Digit Classification (CNN Model):

- **Class Imbalance**: Some digits (e.g., "1" or "0") may appear more frequently, causing the model to favour these classes.
- **Handwriting Bias**: If training data comes from a specific demographic, the model may not generalize well across varied handwriting styles (e.g., children or non-native script writers).

Amazon Product Reviews (NER + Sentiment):

- **Language Bias**: TextBlob may misinterpret informal language, slang, or sarcasm.
- **Brand Bias**: Pretrained spaCy models might over-detect well-known brands like "Amazon" and under-recognize niche or emerging brands.
- **Sentiment Polarity Skew**: TextBlob often classifies moderately positive reviews as "neutral," potentially misrepresenting user satisfaction.

Bias Mitigation Tools & Approaches

TensorFlow Fairness Indicators (for MNIST):

- Can evaluate model performance across different groups (e.g., accuracy per digit).
- Identifies disparities in prediction quality and allows fairness-aware model adjustments.

spaCy Rule-Based Systems (NER):

- Allows addition of domain-specific patterns (e.g., niche product names or local brands).

```
from spacy.matcher import PhraseMatcher
matcher = PhraseMatcher(nlp.vocab)
terms = ["Fire Stick", "Kindle Paperwhite"]
patterns = [nlp.make_doc(text) for text in terms]
matcher.add("PRODUCT", patterns)
doc = nlp("I just bought the new Kindle Paperwhite!")
for match_id, start, end in matcher(doc):
    print("Custom Match:", doc[start:end])
```

General Mitigation Strategies:

- Expand training data to be more representative.
- Include human feedback to refine model outputs.
- Use ensemble methods to cross-validate predictions.

---

2. Troubleshooting Challenge: Fixing Buggy TensorFlow Code

Common Errors in CNN Code:

1. **Missing input channel dimension** in input shape.
2. **Wrong loss function**: using binary_crossentropy for a 10-class classification problem.

Buggy Code Example:

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), input_shape=(28,28), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(10),
])
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Corrected Version:

```
import tensorflow as tf

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), input_shape=(28,28,1), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

-sparse_categorical_crossentropy is appropriate for multi-class classification with integer labels.

-input_shape=(28,28,1) includes the required grayscale channel dimension.

-softmax ensures the final output is interpretable as class probabilities.