Group Members

Lynn Bitok

Adoh Baraza

Salome Wamaitha

Purity Mutunga

Jusper Ageri

Q1: Differences between TensorFlow and PyTorch

Feature	TensorFlow	PyTorch
Computation Graph	Uses *static* (define-and-run) graphs (though TensorFlow 2 supports eager execution).	Uses *dynamic* (define-by-run) graphs, more intuitive for debugging.
Ease of Use	Steeper learning curve; more suited for large-scale deployment.	More Pythonic and user-friendly for research and prototyping.
Deployment	Excellent for production (via TensorFlow Serving, TF Lite, TensorFlow.js).	Deployment options exist (TorchServe, ONNX), but traditionally less extensive.
Ecosystem	Includes Keras API, TensorBoard, TF Hub, and TF Extended (TFX).	Integrated with PyTorch Lightning, TorchVision, and TorchText.

When to choose:

- **TensorFlow** → Large-scale production systems, cross-platform deployment (mobile/web).
- **PyTorch** → Fast research prototyping, easier debugging, dynamic architectures.

Q2: Use cases for Jupyter Notebooks in AI development

1. **Interactive Experimentation & Visualization**
    - Ideal for testing ML models, plotting learning curves, and tuning hyperparameters interactively.
    - Example: Visualizing CNN feature maps or PCA results.
2. **Reproducible Research & Documentation**
    - Combines executable code, results, and markdown explanations in one document.
    - Commonly used for presenting AI experiments or sharing tutorials.

Q3: How spaCy enhances NLP tasks compared to basic Python string operations

- **Linguistic Awareness:**
  spaCy provides tokenization, POS tagging, named entity recognition, and dependency parsing — unlike simple `.split()` or regex operations.

- **Pretrained Models:**
  Includes efficient pretrained pipelines for multiple languages.
- **Performance & Integration:**
  Built in Python for high speed and supports integration with ML frameworks (TensorFlow, PyTorch).

In short: spaCy transforms raw text into structured linguistic data, enabling advanced NLP tasks beyond simple string handling.

---

2. Comparative Analysis: Scikit-learn vs TensorFlow

Aspect	Scikit-learn	TensorFlow
Target Applications	Classical ML: regression, classification, clustering (e.g., SVM, Random Forest).	Deep learning: neural networks, CNNs, RNNs, transformers.
Ease of Use for Beginners	Very beginner-friendly; consistent APIs (`fit()`, `predict()`).	More complex; requires understanding of tensors, layers, and training loops.
Community Support	Mature, strong academic and enterprise adoption; excellent documentation.	Huge global community; widely used in production, research, and industry.

Summary:

- Use **Scikit-learn** for traditional ML tasks and quick prototyping.
- Use **TensorFlow** for deep learning or large-scale neural network models.

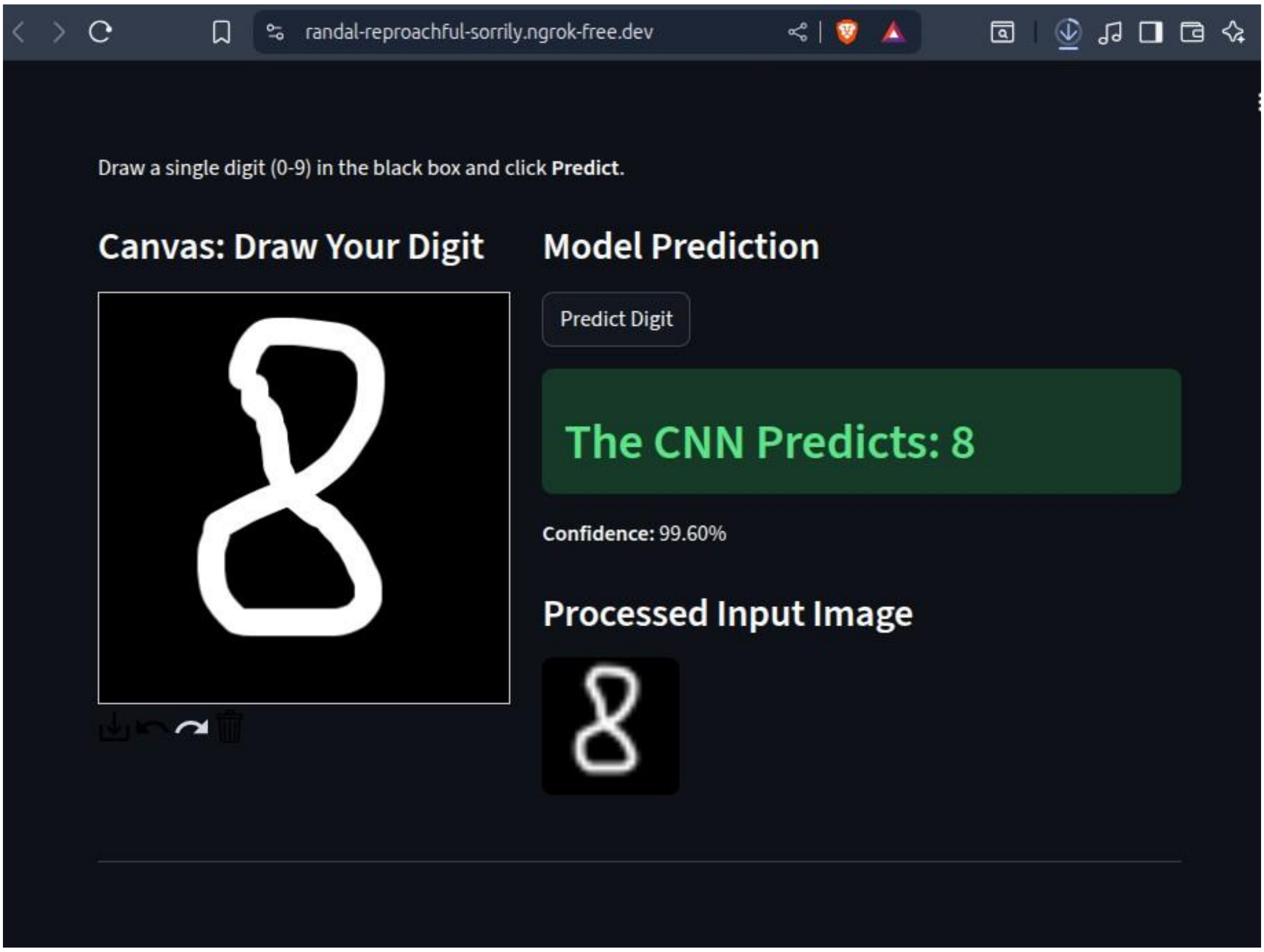
Ethical Considerations

Model	Type of Bias	Description
MNIST (Handwritten Digits)	*Data Representation Bias*	The dataset contains primarily digits written in a particular style (Western handwriting). Models trained on it may perform poorly on digits written by people from different cultures or with disabilities.
	Sampling Bias	MNIST digits are clean and centered; the model might fail on real-world images with noise or rotation.
Amazon Reviews Sentiment Model	*Label Bias*	Sentiment labels may reflect annotators' subjective opinions.
	Linguistic or Demographic Bias	Reviews written in dialects or non-native English may be misclassified (e.g., "This movie was sick!" might be read as negative).
	Imbalance Bias	If positive reviews outnumber negative ones, the model learns to favor "positive" predictions.

Bias Mitigation Using Tools

Tool	How It Helps

Tool	How It Helps
TensorFlow Fairness Indicators	Provides visual reports (like confusion matrices, false positive/negative rates) across *subgroups* (e.g., gender, dialect, region). → Detects whether accuracy is lower for specific demographic groups.
spaCy's Rule-Based Systems	Allows you to enforce *linguistic normalization* before training (e.g., expanding slang, correcting contractions, or balancing named entity detection). → Reduces linguistic bias by ensuring consistent preprocessing.



Draw a single digit (0-9) in the black box and click **Predict**.

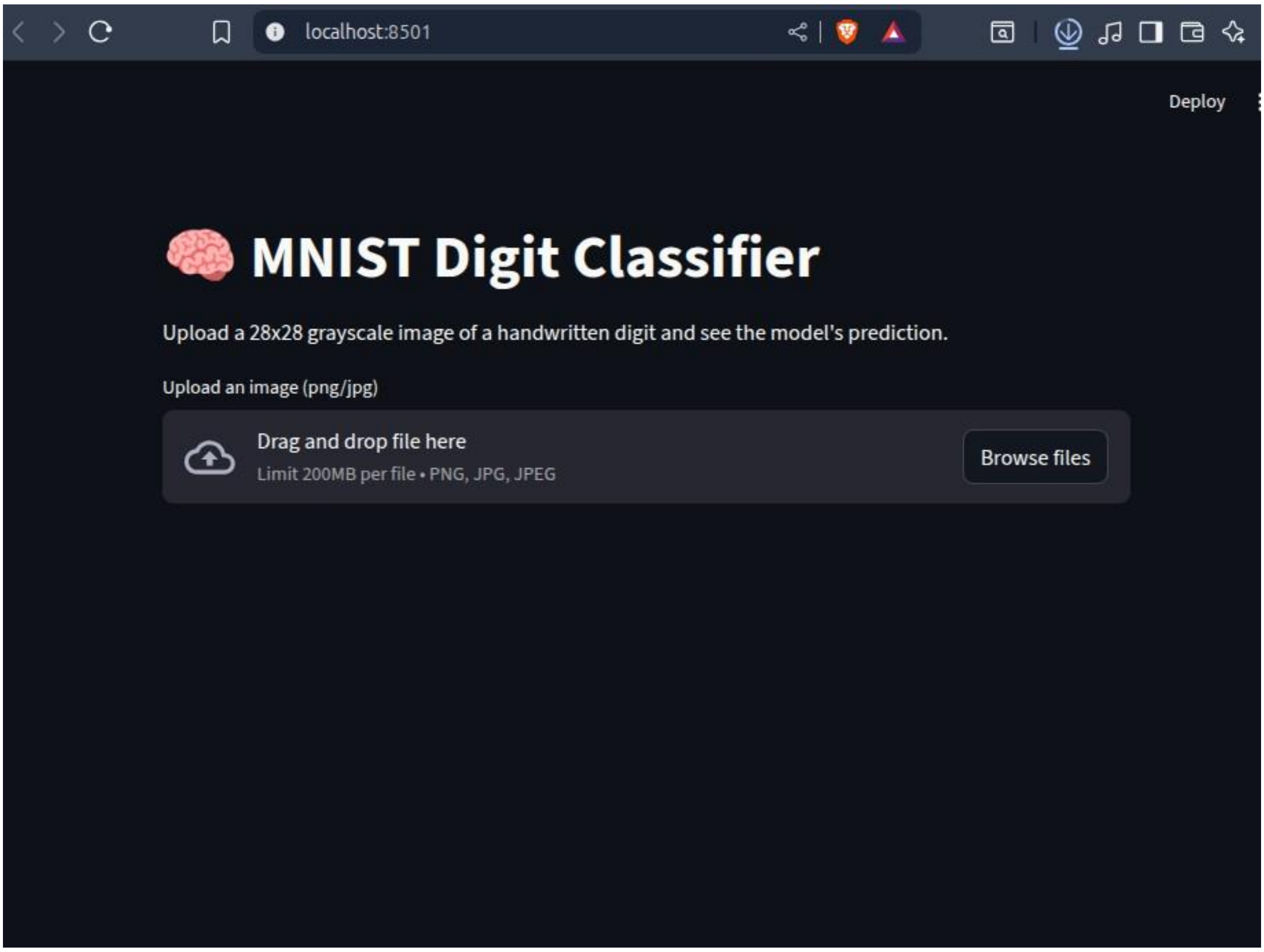
Canvas: Draw Your Digit

Model Prediction

The CNN Predicts: 8

Confidence: 99.60%

Processed Input Image



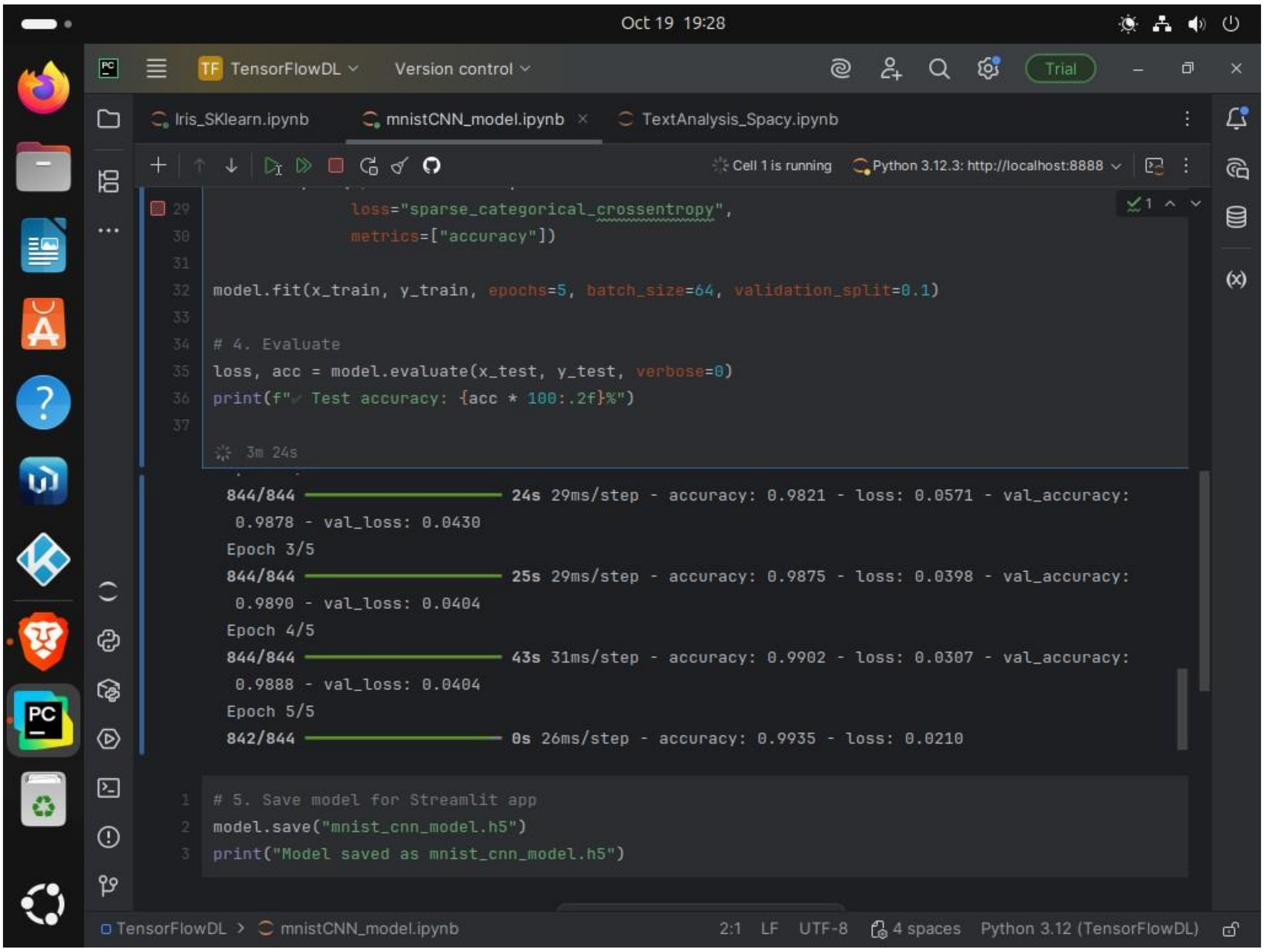
🧠 MNIST Digit Classifier

Upload a 28x28 grayscale image of a handwritten digit and see the model's prediction.

Upload an image (png/jpg)

Drag and drop file here

Limit 200MB per file • PNG, JPG, JPEG



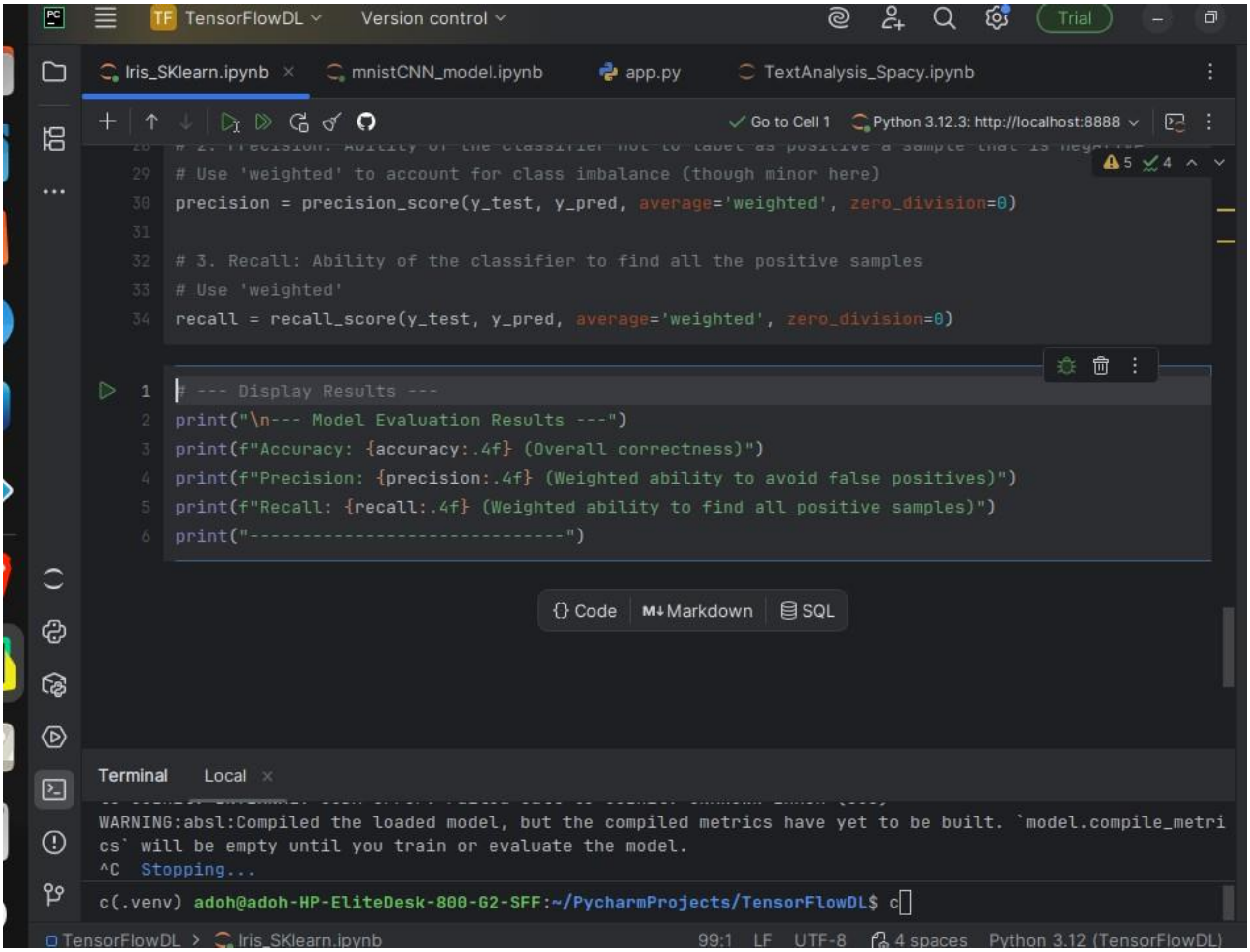
```
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.1)

# 4. Evaluate
loss, acc = model.evaluate(x_test, y_test, verbose=0)
print(f"✓ Test accuracy: {acc * 100:.2f}%")
```

```
844/844 ━━━━━━━━━━━━━━━━━━━━ 24s 29ms/step - accuracy: 0.9821 - loss: 0.0571 - val_accuracy:
 0.9878 - val_loss: 0.0430
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 25s 29ms/step - accuracy: 0.9875 - loss: 0.0398 - val_accuracy:
 0.9890 - val_loss: 0.0404
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 43s 31ms/step - accuracy: 0.9902 - loss: 0.0307 - val_accuracy:
 0.9888 - val_loss: 0.0404
Epoch 5/5
842/844 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.9935 - loss: 0.0210
```

```
# 5. Save model for Streamlit app
model.save("mnist_cnn_model.h5")
print("Model saved as mnist_cnn_model.h5")
```

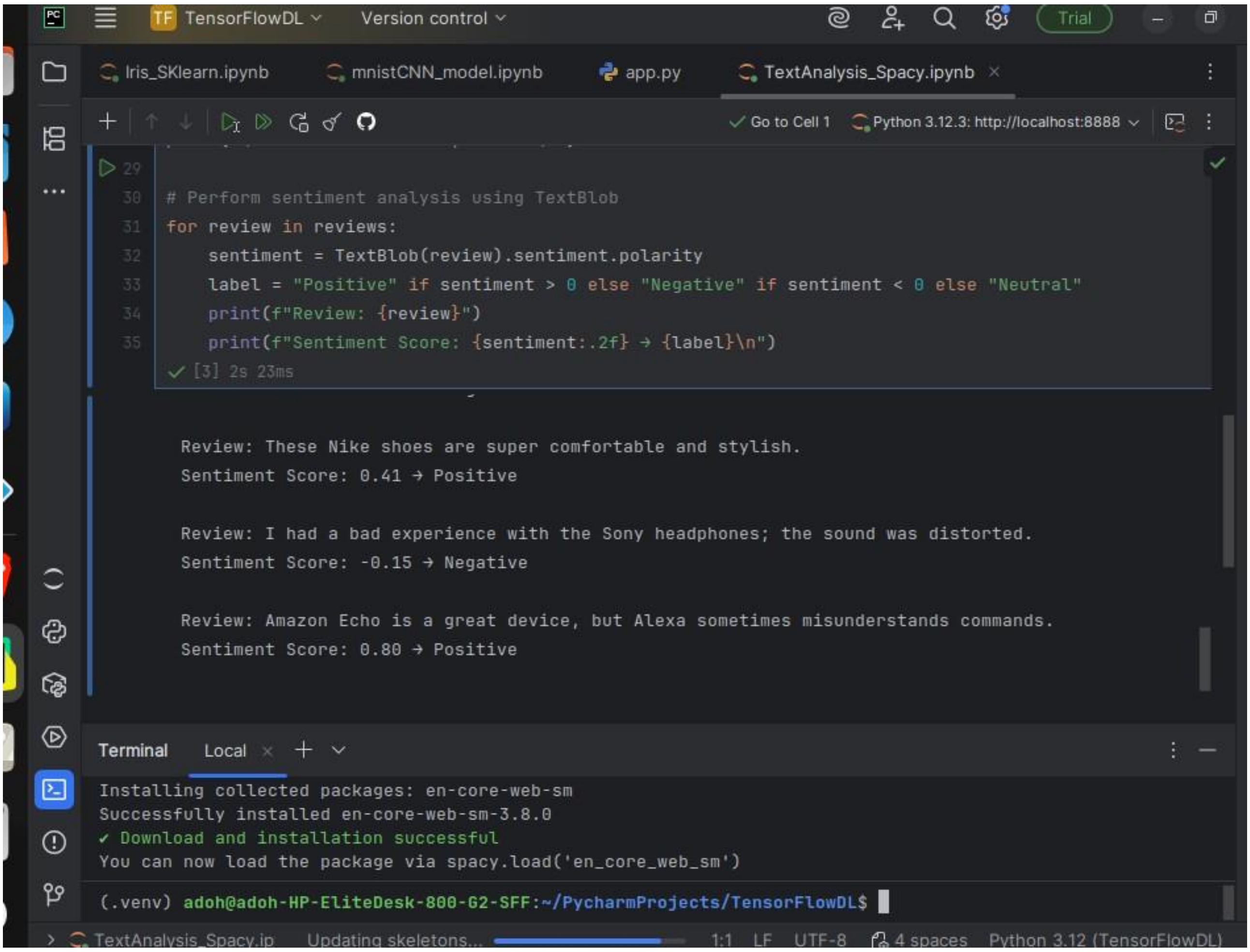


```python
# 2. Precision: Ability of the classifier not to label as positive a sample that is negative
# Use 'weighted' to account for class imbalance (though minor here)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)

# 3. Recall: Ability of the classifier to find all the positive samples
# Use 'weighted'
recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
```

```python
# --- Display Results ---
print("\n--- Model Evaluation Results ---")
print(f"Accuracy: {accuracy:.4f} (Overall correctness)")
print(f"Precision: {precision:.4f} (Weighted ability to avoid false positives)")
print(f"Recall: {recall:.4f} (Weighted ability to find all positive samples)")
print("------------------------------")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metri
cs` will be empty until you train or evaluate the model.
^C  Stopping...
c(.venv) adoh@adoh-HP-EliteDesk-800-G2-SFF:~/PycharmProjects/TensorFlowDL$ c
```



```
# Perform sentiment analysis using TextBlob
for review in reviews:
    sentiment = TextBlob(review).sentiment.polarity
    label = "Positive" if sentiment > 0 else "Negative" if sentiment < 0 else "Neutral"
    print(f"Review: {review}")
    print(f"Sentiment Score: {sentiment:.2f} → {label}\n")
```

```
Review: These Nike shoes are super comfortable and stylish.
Sentiment Score: 0.41 → Positive

Review: I had a bad experience with the Sony headphones; the sound was distorted.
Sentiment Score: -0.15 → Negative

Review: Amazon Echo is a great device, but Alexa sometimes misunderstands commands.
Sentiment Score: 0.80 → Positive
```

```
Installing collected packages: en-core-web-sm
Successfully installed en-core-web-sm-3.8.0
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
(.venv) adoh@adoh-HP-EliteDesk-800-G2-SFF:~/PycharmProjects/TensorFlowDL$
```