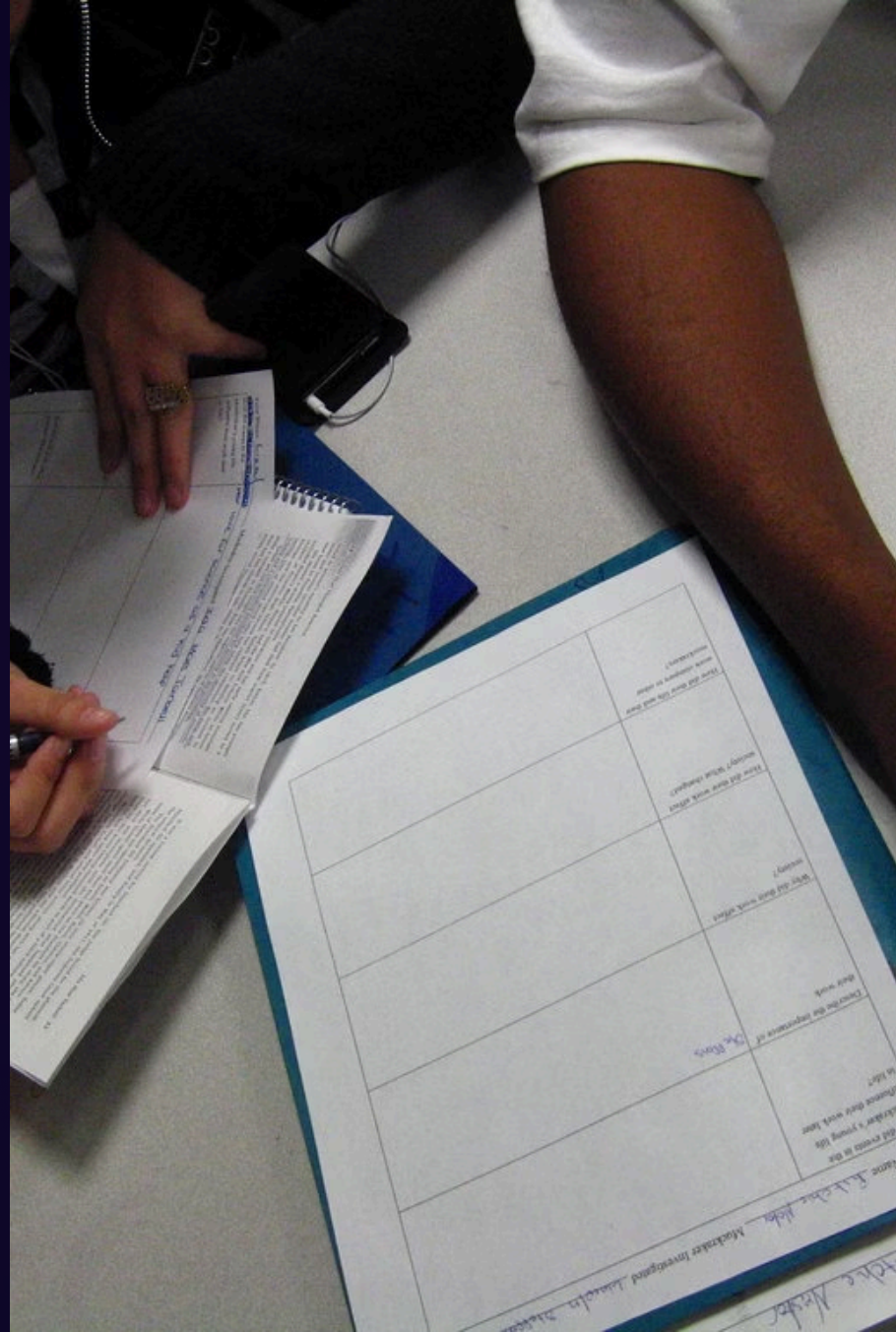
Introduction to Artificial Intelligence — Weeks L1 & L2

This two-week lecture (L1 + L2) covers foundational theory in artificial intelligence (AI). We'll define what AI is, highlight key differences between related terms (machine learning, deep learning, automation), give clear, beginner-friendly explanations, show practical code examples you can run, and include real photos to connect ideas to real-world tools and people. By the end you'll understand core concepts and be able to read simple AI code and reason about when to use different approaches.

Theme colors used across slides for emphasis: #876cd4ff, #D783D8, #FF90A5, #FFB071, and #14083A.



What is Artificial Intelligence?

Definition: **Artificial Intelligence (AI)** is the field of computer science that builds systems able to perform tasks that normally require human intelligence, such as reasoning, perception, decision-making, and natural language understanding.

Long-form explanation (beginner-friendly): AI systems use algorithms and data to make predictions or take actions. Some systems follow fixed rules; others learn patterns from examples. AI is not one single technology — it's a collection of methods and ideas aimed at making machines useful partners for people.

Quick real-world examples: voice assistants (speech → command), email spam filtering (classify messages), image search (find photos that match a query), and recommendation systems (suggest products or movies).

Tip: Think of AI as "smart software" that either follows rules or learns from data — we'll explore both approaches.

Related Terms: Machine Learning & Deep Learning

Machine Learning (ML)

Definition: **Machine Learning** is a subset of AI where systems improve performance by learning patterns from data instead of using only explicit rules. Key idea: provide examples (data) and the algorithm extracts patterns to make predictions.

Simple ML code example (Python, scikit-learn):

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
print(clf.score(X_test, y_test))
```

Deep Learning (DL)

Definition: **Deep Learning** is a family of ML methods that use multi-layer neural networks to learn complex patterns from very large datasets. It's especially powerful for images, audio, and language.

Simple DL code example (Python, TensorFlow/Keras):

```
from tensorflow import keras
model = keras.Sequential([
  keras.layers.Dense(64, activation='relu', input_shape=(input_dim,)),
  keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
model.fit(train_X, train_y, epochs=10, batch_size=32)
```

Relationship: AI ⊃ ML ⊃ DL. Use ML for structured data and simpler problems; use DL when you have lots of data and complex patterns (images, speech).

Automation vs AI — What's the Difference?

Definition: **Automation** executes pre-defined tasks without learning. It follows explicit rules. **AI** may automate tasks too, but it can adapt based on data and predictions.

Example contrast: - Automation: a script that renames files by fixed rules — predictable, no learning. - AI: a model that reads invoices and learns to extract amounts from varied layouts — learned behavior from examples.

1

When to use Automation

Repeatable, rule-based tasks with predictable inputs and no need to improve from examples.

2

When to use AI

Tasks with variability, ambiguity, or patterns that are hard to express as rules (e.g., image recognition).

Code example (simple automation vs learning):

```
# Automation: deterministic text replace
text = text.replace("USD", "$")

# AI (ML): find if sentence is positive or negative with a tiny trained model
pred = clf.predict([vectorize(sentence)])
```

Core Concepts: Supervised, Unsupervised, and Reinforcement Learning

Definitions (simple): - **Supervised learning**: model learns from labeled examples (input → correct output). Used for classification and regression. - **Unsupervised learning**: model finds structure in unlabeled data (clusters, patterns). - **Reinforcement learning**: an agent learns by trial and error, receiving rewards or penalties for actions.

Supervised

Labelled data; predict outcomes (e.g., spam detection).

Unsupervised

No labels; find groups or reduce dimensions (e.g., customer segments).

Reinforcement

Learn actions to maximize reward (e.g., game-playing agents, robotics).

Short code snippets:

```
# Supervised (classification) - scikit-learn
clf.fit(X_train, y_train)

# Unsupervised (k-means clustering)
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3).fit(X)

# Reinforcement (pseudo-code)
for episode in episodes:
    action = policy(state)
    next_state, reward = env.step(action)
    update_policy(state, action, reward, next_state)
    state = next_state
```

Data: The Fuel of AI

Why data matters: Models learn patterns from data — more relevant, high-quality data generally produces better performance. Poor or biased data leads to wrong or unfair results.

Key data concepts for beginners: - Labels: Correct answers used in supervised learning. - Features: Input attributes the model uses (age, pixels, words). - Train / Validation / Test split: separate data to train, tune, and evaluate models fairly. - Overfitting vs Underfitting: overfitting memorizes training data (bad), underfitting fails to capture patterns (also bad).

Best practices

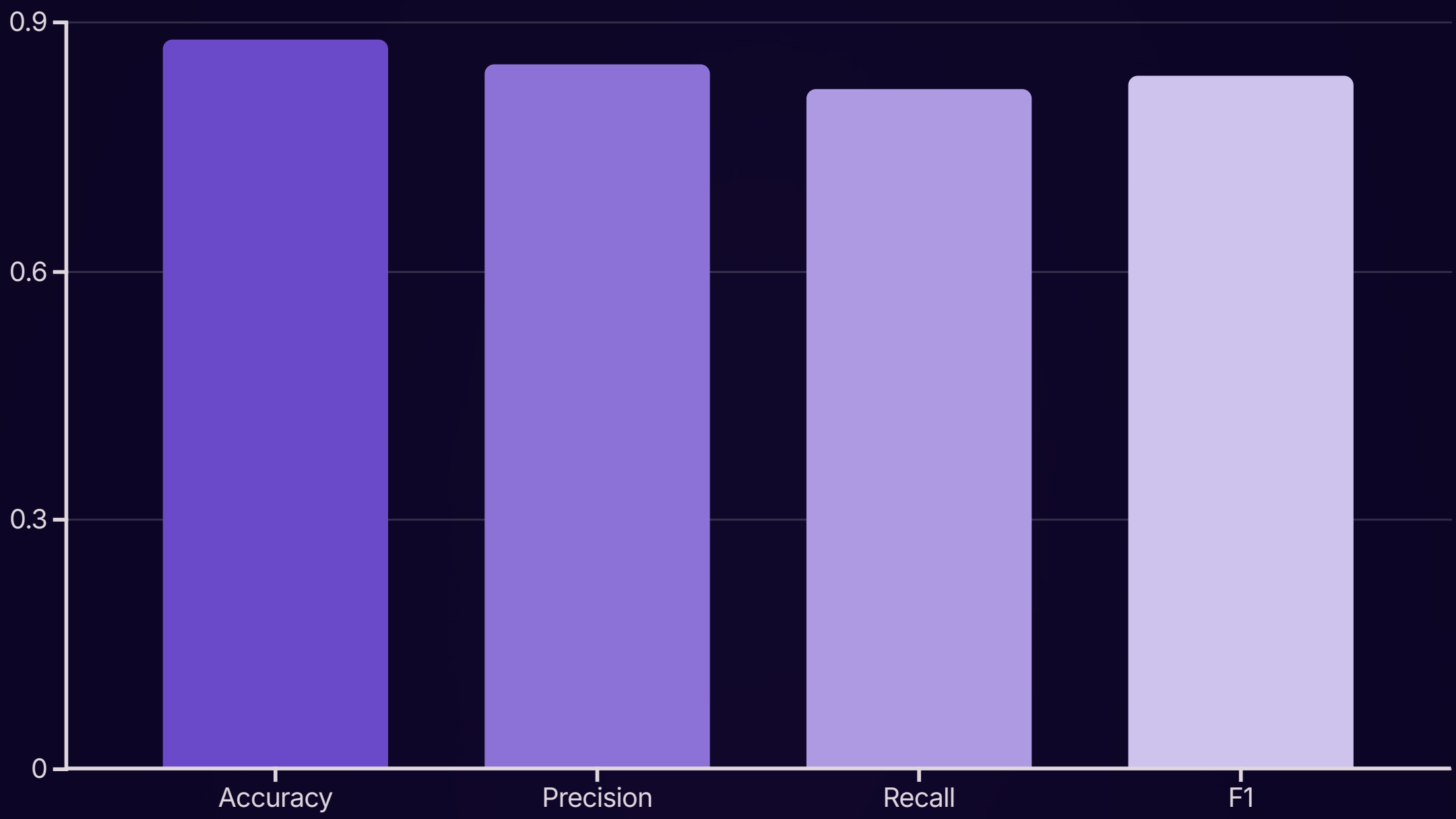
- Clean and normalize data
- Remove or correct obvious errors
- Ensure class balance when needed
- Document data sources and limitations

Simple code: split data

```
from sklearn.model_selection import train_test_split
X_tr, X_temp, y_tr, y_temp = train_test_split(X, y, test_size=0.3)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5)
```

Model Evaluation — How We Know a Model Works

Definitions (practical): - **Accuracy**: proportion of correct predictions (works for balanced classes). - **Precision**: fraction of positive predictions that are correct (important when false positives are costly). - **Recall**: fraction of actual positives detected (important when missing positives is costly). - **F1 score**: harmonic mean of precision and recall — useful balance for class-imbalanced problems.



Code to compute metrics:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
y_pred = clf.predict(X_test)
print(accuracy_score(y_test, y_pred))
print(precision_score(y_test, y_pred))
print(recall_score(y_test, y_pred))
print(f1_score(y_test, y_pred))
```

Interpretability & Ethics — Responsible AI Basics

Definitions: - **Interpretability**: understanding why a model makes a prediction. Simple models (decision trees, linear regression) are more interpretable than large neural networks. - **Bias**: when model outputs systematically disadvantage certain groups. Bias often comes from biased data or poor design choices. - **Fairness & Transparency**: designing systems to treat people fairly, explain decisions, and allow accountability.

Fairness

Check for unequal performance across groups and mitigate with balanced data or fairness-aware techniques.

Explainability

Use interpretable models or tools (SHAP, LIME) to explain predictions.

Privacy

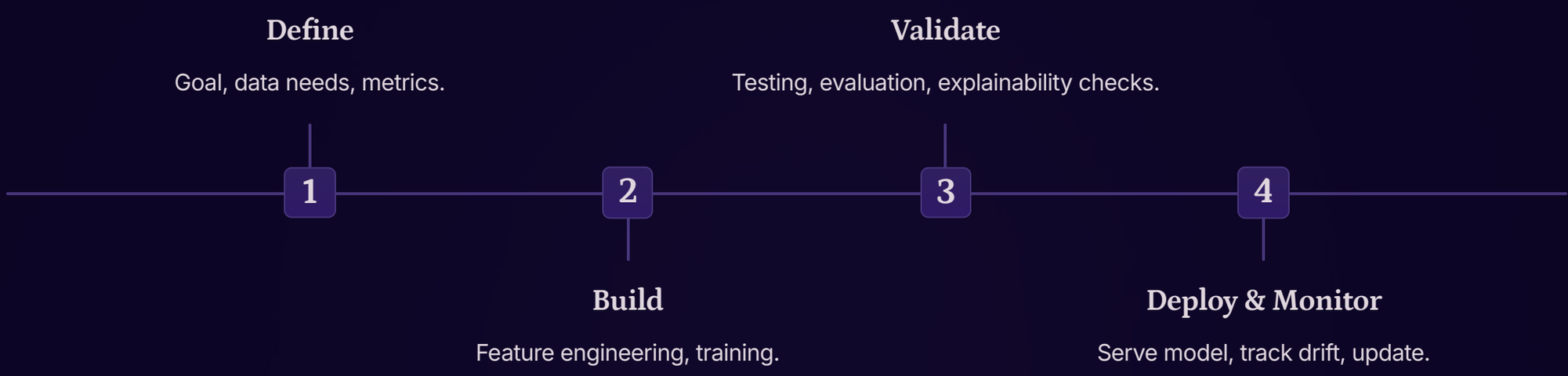
Protect sensitive data; apply anonymization, encryption, and minimal data collection.

Example code (using SHAP for a tree model explanation):

```
import shap
explainer = shap.TreeExplainer(clf)
shap_values = explainer.shap_values(X_sample)
shap.summary_plot(shap_values, X_sample)
```

Practical Workflow: From Problem to Deployment

A practical step-by-step workflow: 1. Define problem and success criteria (clear labels, measurable metrics). 2. Gather and explore data (visualize, clean, label). 3. Choose model family and baseline model (start simple). 4. Train and validate (tune hyperparameters). 5. Evaluate on hold-out test set and check fairness and robustness. 6. Deploy model and monitor performance; retrain when data distribution shifts.



Minimal deployment code example (Flask API to serve a scikit-learn model):

```
from flask import Flask, request, jsonify
import joblib
model = joblib.load('model.pkl')
app = Flask(__name__)
@app.route('/predict', methods=['POST'])
def predict():
    data = request.json['features']
    pred = model.predict([data])
    return jsonify({'prediction': int(pred[0])})
```

Key Takeaways, Next Steps & Resources

Summary (clear bullets): - AI means systems that perform tasks requiring intelligence; ML and DL are subsets used for learning from data. - Choose methods based on problem complexity, data size, and need for interpretability. - Data quality, evaluation metrics, and ethical considerations are as important as model choice. - Start simple, iterate, evaluate fairly, and monitor models after deployment.

Hands-on practice

Try small projects: Titanic classifier, MNIST digit recognition, or sentiment analysis on tweets.

Learn more

Recommended free resources: Coursera AI for Everyone (Andrew Ng), fast.ai practical deep learning, scikit-learn tutorials.

Next steps

Build a portfolio project, document data and decisions, and practice explaining model behavior to non-technical people.

Final code exercise (simple end-to-end starter): load CSV, preprocess, train a logistic regression, evaluate, and save model. Use the earlier snippets combined. If you want, I can generate that full notebook code next.

Ready for Week 3? We can dive into neural network internals, convolutional networks for images, and practical debugging techniques.