Week 7: Building Your First ML Models

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra

IIT Gandhinagar

Part 1: The Big Picture

What does it mean to "build" an ML model?

Remember Our Netflix Journey?

```
Week 1: Collected movie data (APIs, scraping)
Week 2: Cleaned and organized it (Pandas)
Week 3: Labeled movie success/failure (annotation)
Week 4: Made labeling efficient (active learning)
Week 5: Got more data (augmentation)
Week 6: Used LLMs to help (APIs)
        ↓
Week 7: NOW WE BUILD THE MODEL!
🎉
```

We finally have good data. Time to predict!

What Are We Predicting?

Our Netflix Problem:

Given movie features → Predict if it will be successful

```
INPUT (What we know)              OUTPUT (What we predict)
─────────────────────             ────────────────────────

• Genre: Action
• Budget: $150M                    → SUCCESS or FAILURE?
• Director: Nolan
• Runtime: 148 mins
```

This is called **Classification** (putting things in categories)

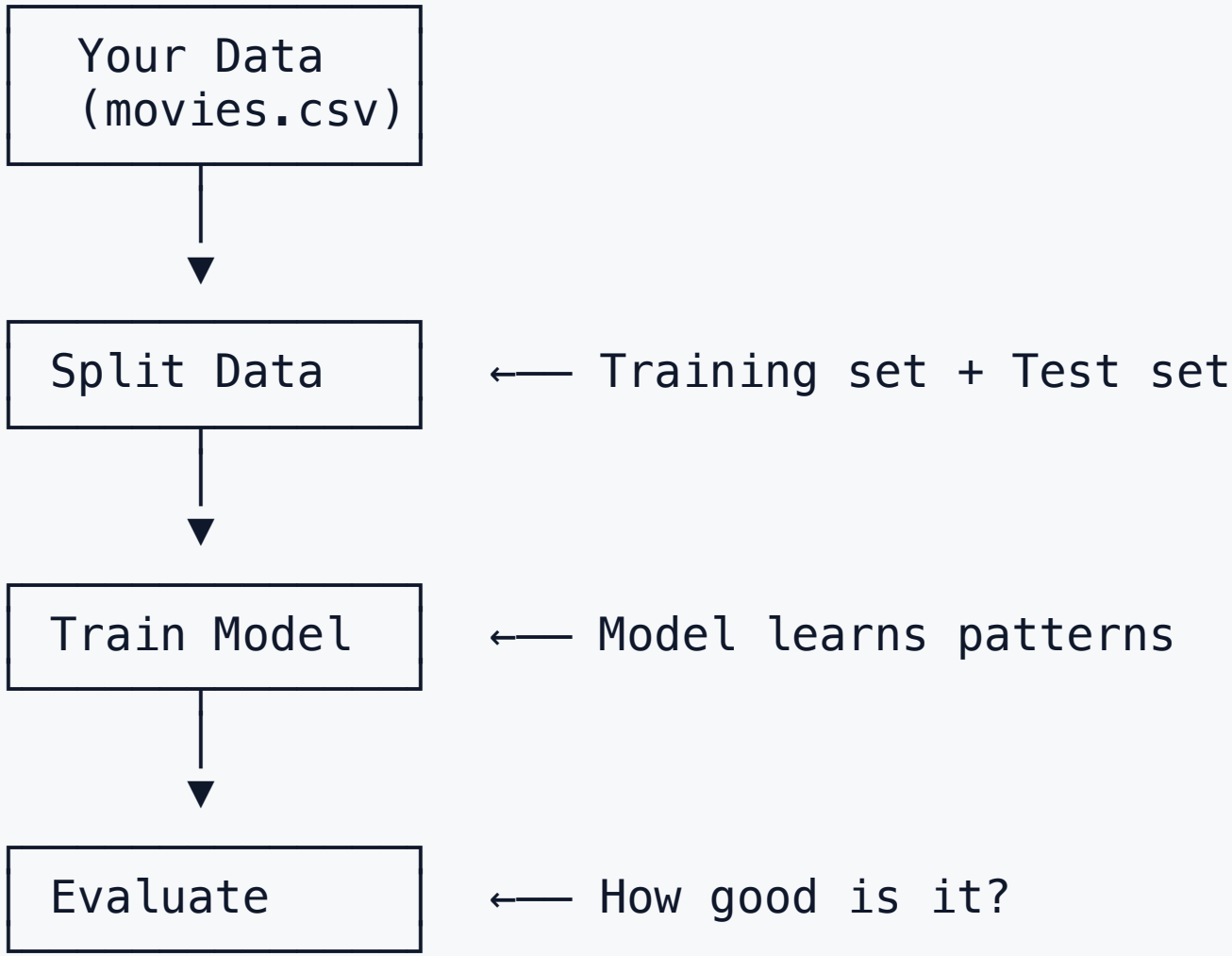
Two Types of Predictions

```
      CLASSIFICATION

  Predict a CATEGORY

  • Success / Failure
  • Spam / Not Spam
  • Cat / Dog / Bird

  "Which box does this go in?"
```

```
        REGRESSION

  Predict a NUMBER

  • $500M revenue
  • 7.5 rating
  • 25°C temperature

  "How much / How many?"
```

Today: We'll focus on classification (predicting movie success)

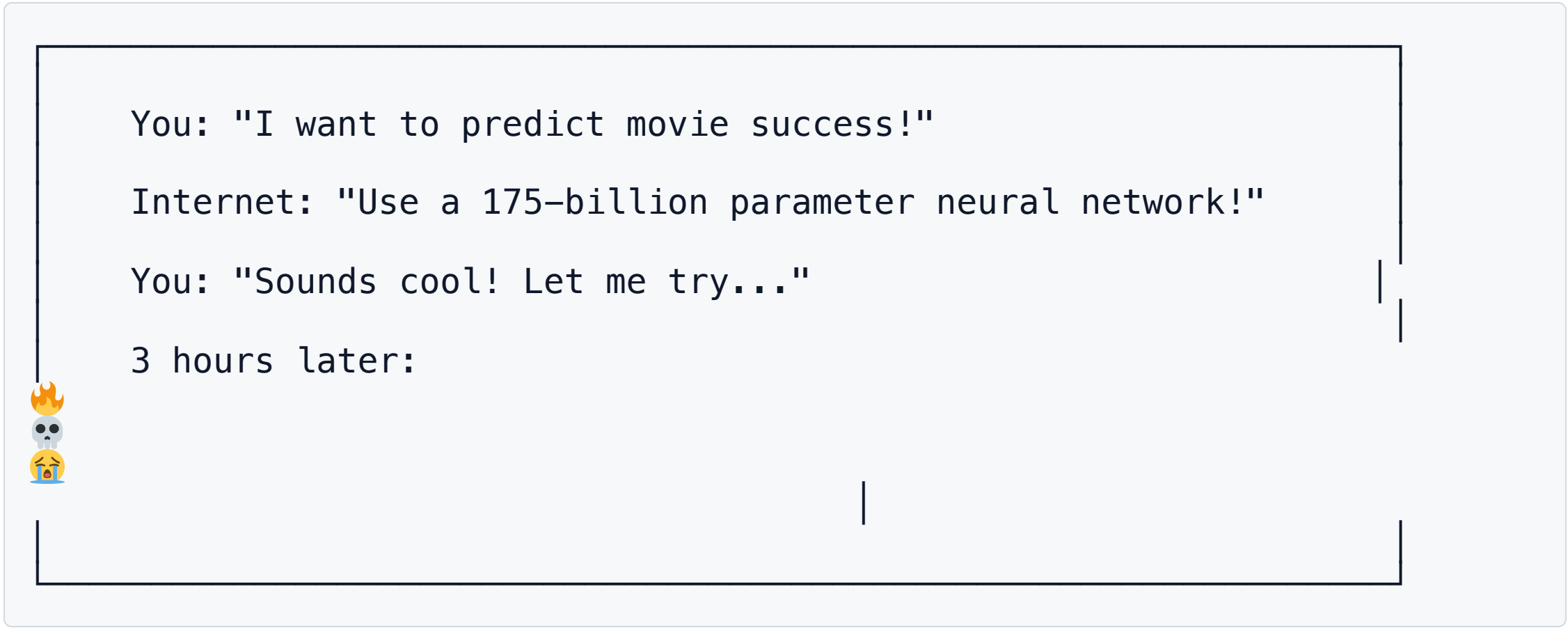
The ML Workflow (Simple Version)



Part 2: Starting Simple - Baseline Models

Why you should never start with deep learning

The Temptation



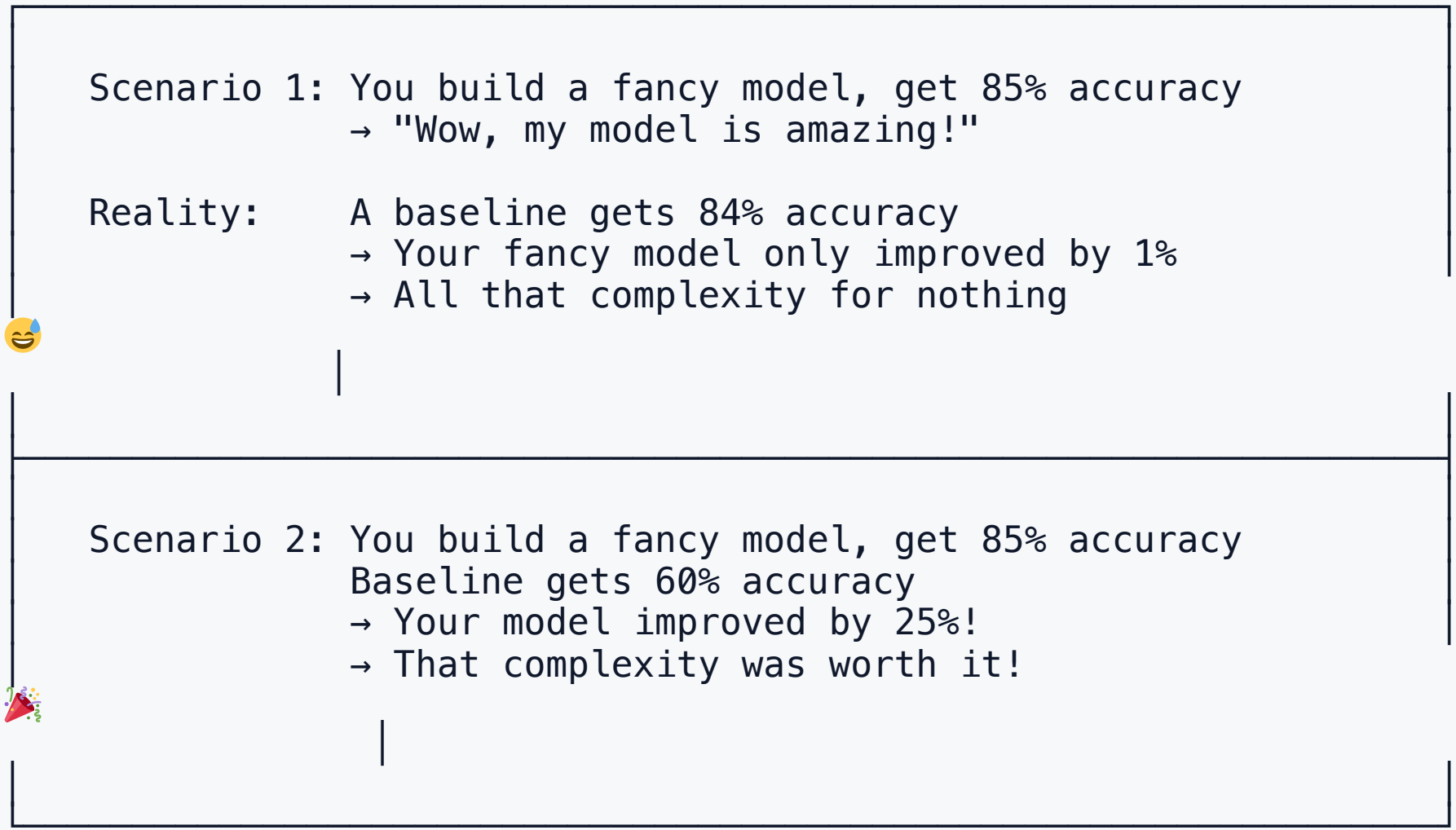
DON'T DO THIS!

What is a Baseline?

A **baseline** is the simplest possible solution that works.

```
                    BASELINE EXAMPLES

Task: Predict if movie succeeds

Dumb Baseline: "Just predict the most common outcome"
               If 70% of movies succeed → always say SUCCESS
               Accuracy: 70% (for free!)

Simple Model:  Logistic Regression
               (One line of code, 80% accuracy?)

Complex Model: Deep Neural Network
               (1000 lines of code, 82% accuracy?)
```

Why Baselines Matter



The Simplest Baseline: "Just Guess"

```
# The dumbest model possible
def dumb_predictor(movie):
    return "SUCCESS"  # Always predict success

# If 70% of movies succeed, this gets 70% accuracy!
```

This is called a "Majority Class Classifier"

```
from sklearn.dummy import DummyClassifier

# Create the dumbest possible classifier
baseline = DummyClassifier(strategy='most_frequent')
baseline.fit(X_train, y_train)

accuracy = baseline.score(X_test, y_test)
print(f"Dumb baseline accuracy: {accuracy:.1%}")
```


Any real model must beat this!

Baseline Model 1: Logistic Regression

Think of it as: A weighing scale for features

```
┌──────────────────────────────────────────────────────────────────────┐
│                                                                      │
│  Feature              Weight      Value       Contribution           │
│  ─────────────────    ──────      ─────       ────────────           │
│  Budget ($M)          +0.3        150         +45                    │
│  Star Power           +0.5        8           +4                     │
│  Is Sequel            +0.2        1           +0.2                   │
│  Is January Release   −0.4        0           0                      │
│                                               ─────                  │
│                                   Total:      +49.2                  │
│                                                                      │
│  If Total > 0 → Predict SUCCESS                                      │
│  If Total < 0 → Predict FAILURE                                      │
│                                                                      │
└──────────────────────────────────────────────────────────────────────┘
```


It just adds up weighted features!

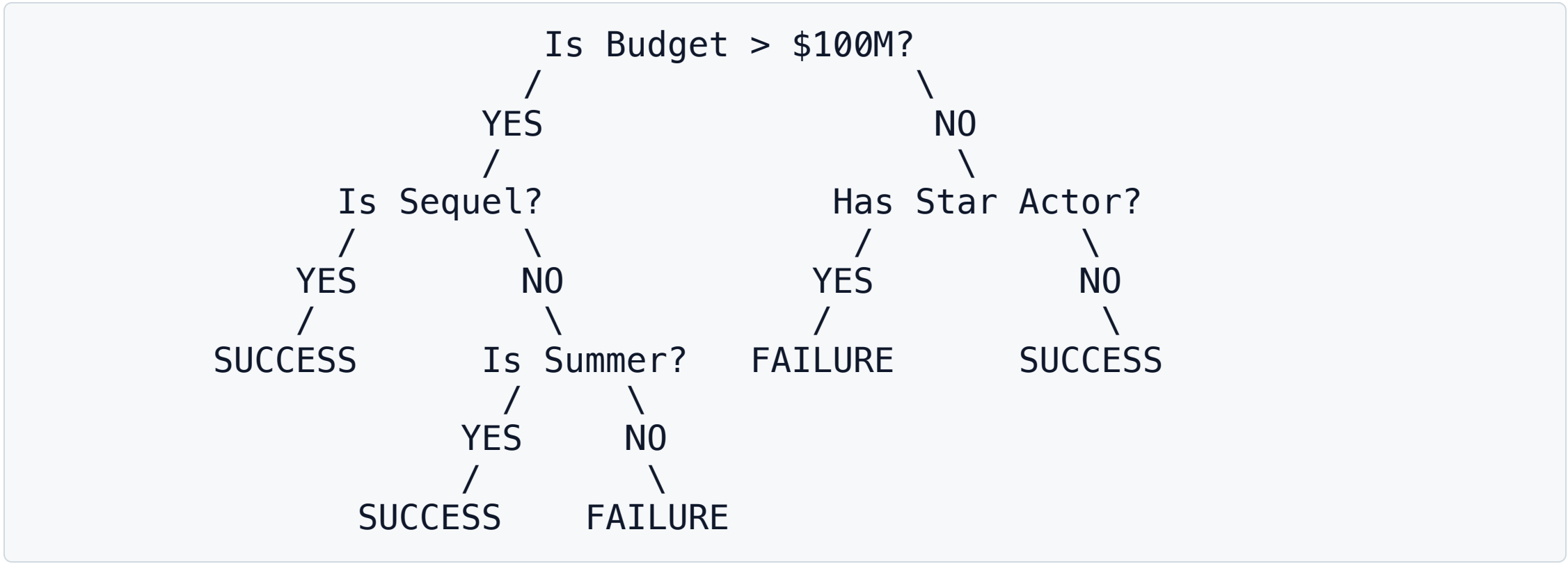
Logistic Regression in Code

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    features, labels, test_size=0.2, random_state=42
)

# Create and train the model (2 lines!)
model = LogisticRegression()
model.fit(X_train, y_train)

# Evaluate
accuracy = model.score(X_test, y_test)
print(f"Logistic Regression accuracy: {accuracy:.1%}")
```

That's it! A working ML model in 4 lines.

Baseline Model 2: Decision Tree

Think of it as: A flowchart of yes/no questions



Humans can actually read and understand this!

Decision Tree in Code

```
from sklearn.tree import DecisionTreeClassifier

# Create and train
tree = DecisionTreeClassifier(max_depth=5)  # Don't go too deep!
tree.fit(X_train, y_train)

# Evaluate
accuracy = tree.score(X_test, y_test)
print(f"Decision Tree accuracy: {accuracy:.1%}")
```

You can even visualize it:

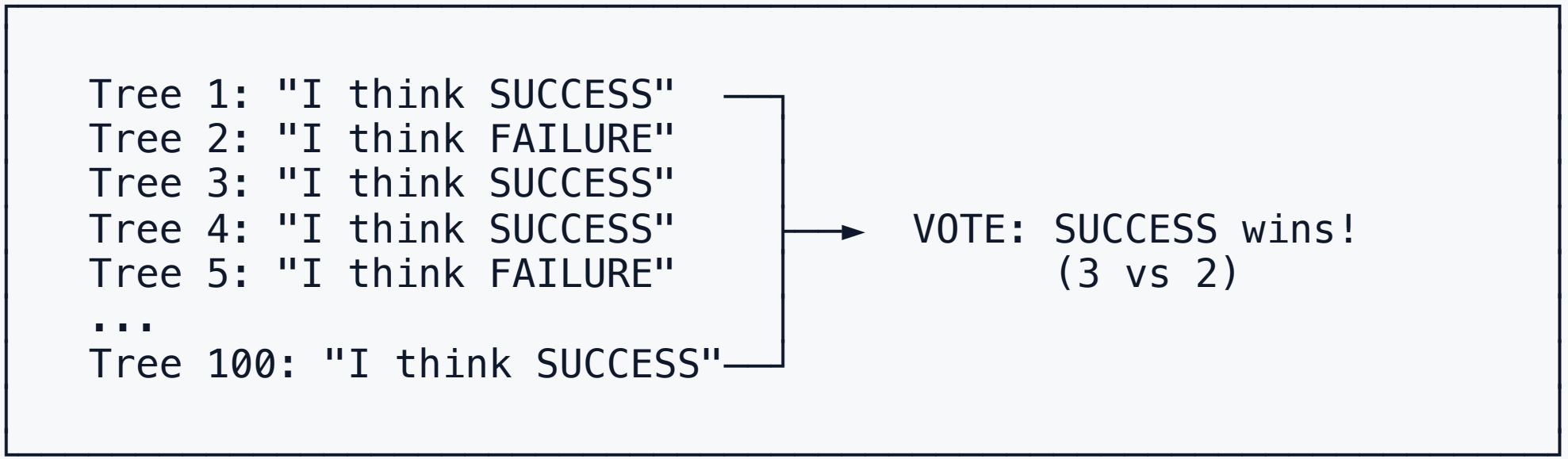
```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 10))
plot_tree(tree, feature_names=feature_names, filled=True)
plt.show()
```

Which Baseline to Use?

BASELINE SELECTION GUIDE	
Your Situation	Recommended Baseline
Just starting	Logistic Regression (fast, simple, often works well)
Need interpretability (explain to your boss)	Decision Tree (you can see the rules)
Mixed data types (numbers + categories)	Random Forest (handles everything)
Want best performance (don't care how)	AutoML (we'll learn this later!)

Baseline Model 3: Random Forest

Think of it as: Asking 100 decision trees and taking a vote



Wisdom of crowds: Many weak learners → One strong learner

Random Forest in Code

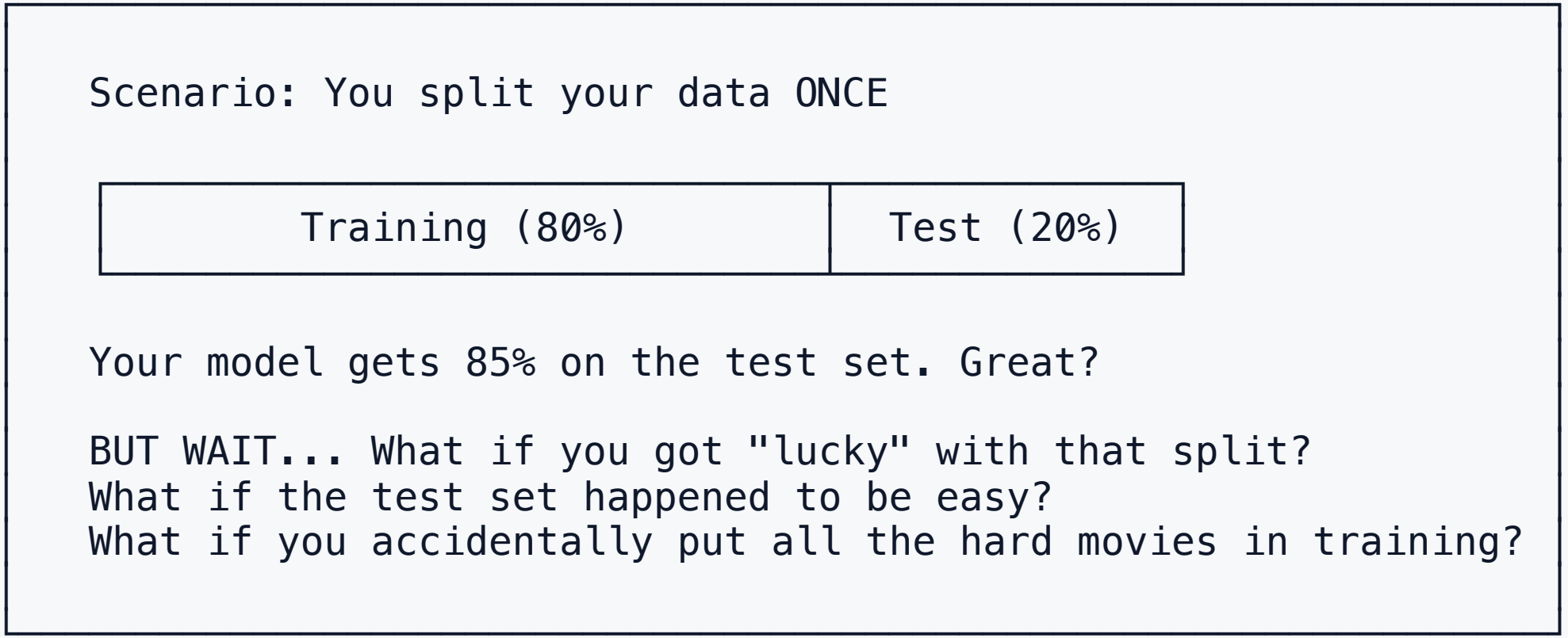
```
from sklearn.ensemble import RandomForestClassifier

# Create and train
forest = RandomForestClassifier(n_estimators=100, random_state=42)
forest.fit(X_train, y_train)

# Evaluate
accuracy = forest.score(X_test, y_test)
print(f"Random Forest accuracy: {accuracy:.1%}")
```

Often the best simple model! Very hard to beat.

Part 3: Cross-Validation

How to really know if your model is good

The Problem with One Test Set



One test set = One roll of the dice

The Solution: Cross-Validation

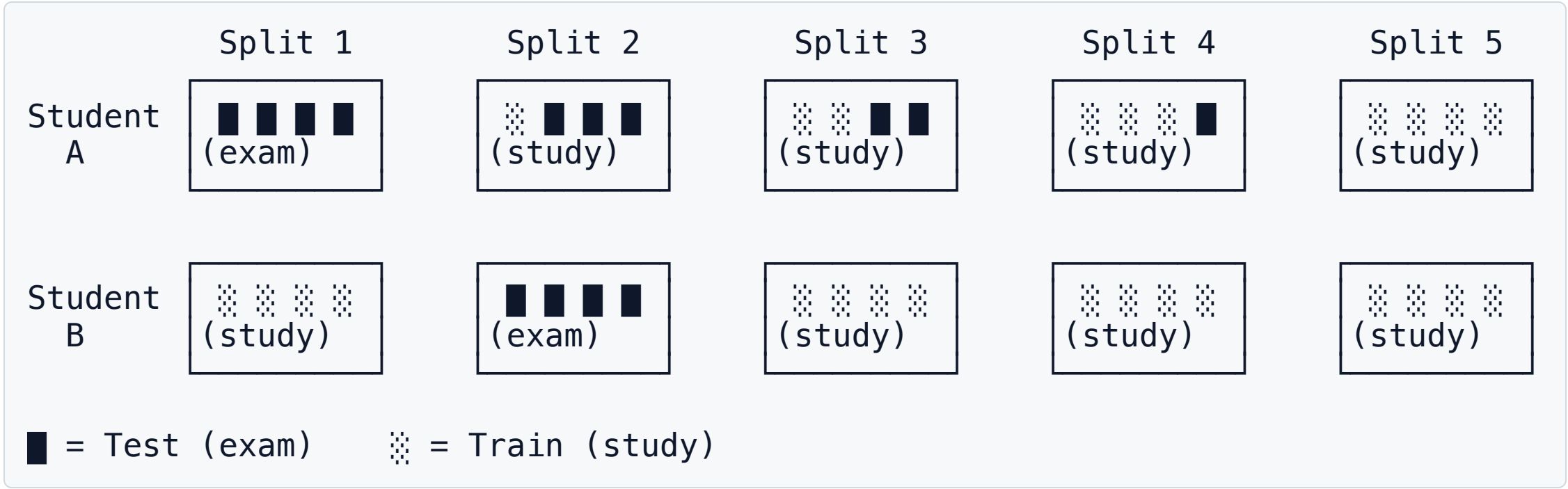
Idea: Test on EVERY part of your data (not just 20%)

```
                    5-FOLD CROSS-VALIDATION

Fold 1: [TEST][Train][Train][Train][Train]  → Accuracy: 82%
Fold 2: [Train][TEST][Train][Train][Train]  → Accuracy: 85%
Fold 3: [Train][Train][TEST][Train][Train]  → Accuracy: 84%
Fold 4: [Train][Train][Train][TEST][Train]  → Accuracy: 81%
Fold 5: [Train][Train][Train][Train][TEST]  → Accuracy: 83%

Average: 83% ± 1.5%
```

Now we know: "My model gets ~83% accuracy, give or take 1.5%"

Cross-Validation: Visual Intuition

Think of it like a **rotating exam schedule**:



Every data point gets tested exactly once!

Cross-Validation in Code

```
from sklearn.model_selection import cross_val_score

# Create model
model = RandomForestClassifier(n_estimators=100)

# Run 5-fold cross-validation
scores = cross_val_score(model, X, y, cv=5)

print(f"Scores for each fold: {scores}")
print(f"Average accuracy: {scores.mean():.1%}")
print(f"Standard deviation: {scores.std():.1%}")
```

Output:

```
Scores for each fold: [0.82, 0.85, 0.84, 0.81, 0.83]
Average accuracy: 83.0%
Standard deviation: 1.5%
```

Why Cross-Validation Matters

MODEL COMPARISON		
Model	Single Test	5-Fold CV
Logistic Reg.	78%	76% ± 2%
Decision Tree	82%	75% ± 5% ← High variance!
Random Forest	84%	83% ± 1% ← Most stable!

Insights:

- Decision Tree looked good on one test, but it's unstable
- Random Forest is not only accurate but **consistent**

Cross-validation reveals the truth!

Quick Summary So Far

```
                    WHAT WE LEARNED

1. BASELINES: Always start simple
   • Majority classifier (the dumbest possible)
   • Logistic Regression (weighted sum of features)
   • Decision Tree (flowchart of rules)
   • Random Forest (voting committee of trees)

2. CROSS-VALIDATION: Test on all your data
   • Split data into 5 (or 10) folds
   • Each fold takes a turn being the test set
   • Get average ± standard deviation
   • Much more reliable than a single test set
```

Part 4: AutoML - Let the Computer Do It

The lazy (smart) way to build models

The Problem with Manual ML

```
┌──────────────────────────────────────────────────────────────────────────┐
│                        BUILDING ML MODELS MANUALLY                       │
└──────────────────────────────────────────────────────────────────────────┘

  1. Try Logistic Regression... okay
  2. Try Decision Tree... not great
  3. Try Random Forest... better
  4. Try XGBoost... hmm, similar
  5. Try Neural Network... takes forever
  6. Tune hyperparameters for each...
  7. Try different feature combinations...
  8. Repeat steps 1-7 many times...

  Time spent: 3 days
  Hair remaining: None
```


There has to be a better way!

Enter AutoML

AutoML = Automatic Machine Learning

```
You: "Here's my data. Give me the best model."

AutoML: "On it! Let me try 50 different models,
         tune their parameters, combine the best ones,
         and give you a super-ensemble."

You: *goes to get coffee*

AutoML: "Done! Here's a model with 87% accuracy."
```

This is not magic. It just automates what experts do manually.

AutoGluon: AutoML Made Easy

AutoGluon (by Amazon) is one of the best AutoML tools.

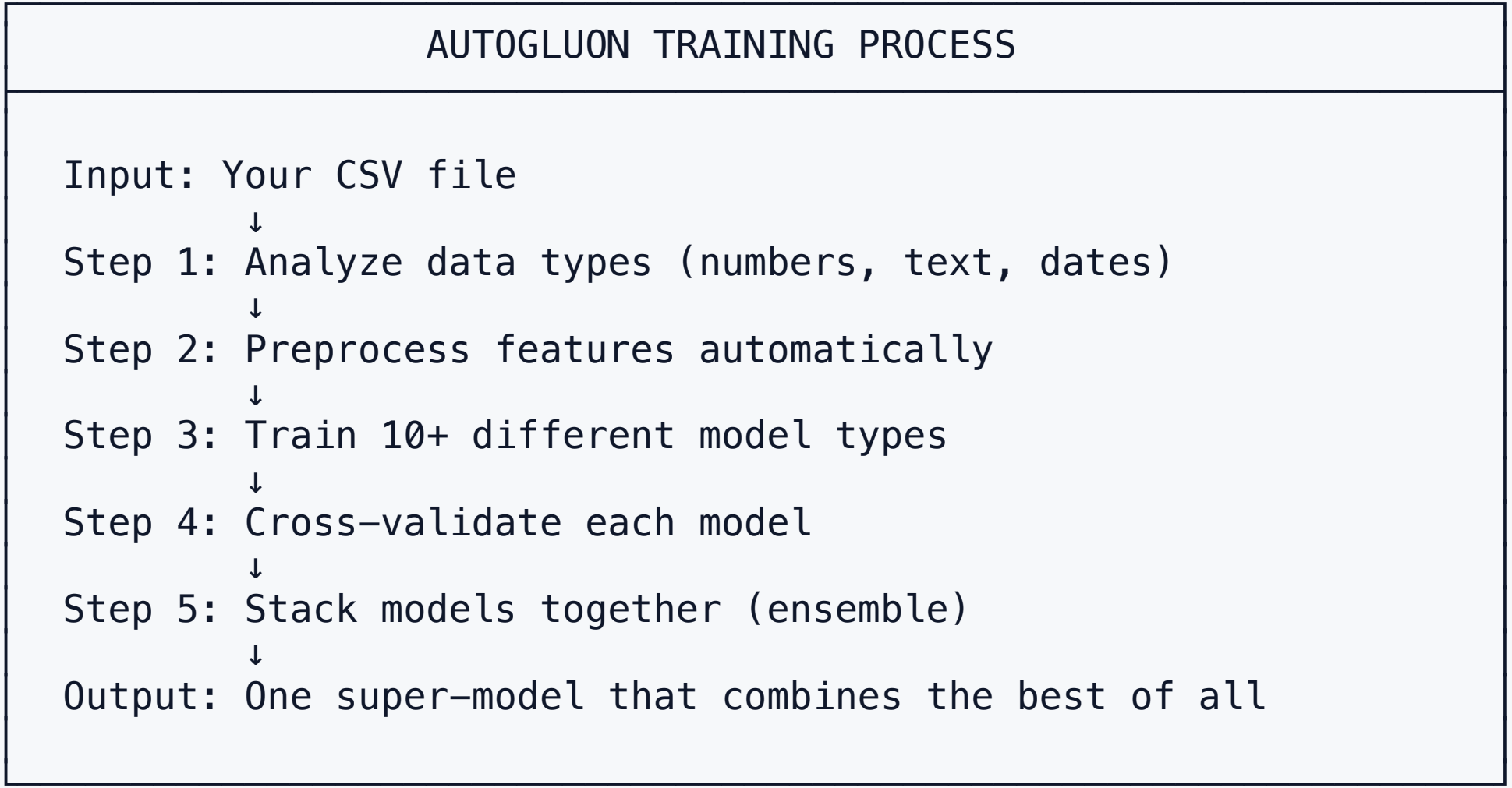
```
┌──────────────────────────────────────────────────────────────────────┐
│                        WHAT AUTOGLUON DOES                           │
├──────────────────────────────────────────────────────────────────────┤
│                                                                      │
│  1. Automatically handles missing values                             │
│  2. Automatically encodes categorical features                       │
│  3. Trains multiple model types:                                     │
│     • Random Forest                                                  │
│     • XGBoost, LightGBM, CatBoost (gradient boosting)                │
│     • Neural Networks                                                │
│     • And more...                                                    │
│  4. Tunes hyperparameters                                            │
│  5. Creates an ensemble of the best models                           │
│  6. Uses cross-validation internally                                 │
│                                                                      │
└──────────────────────────────────────────────────────────────────────┘
```

AutoGluon in 3 Lines of Code

```
from autogluon.tabular import TabularPredictor

# Step 1: Create the predictor
predictor = TabularPredictor(label='success')

# Step 2: Train on your data (that's it!)
predictor.fit(train_data)

# Step 3: Make predictions
predictions = predictor.predict(test_data)
```

Seriously. That's the entire code.

What Happens Inside AutoGluon?



AutoGluon Leaderboard

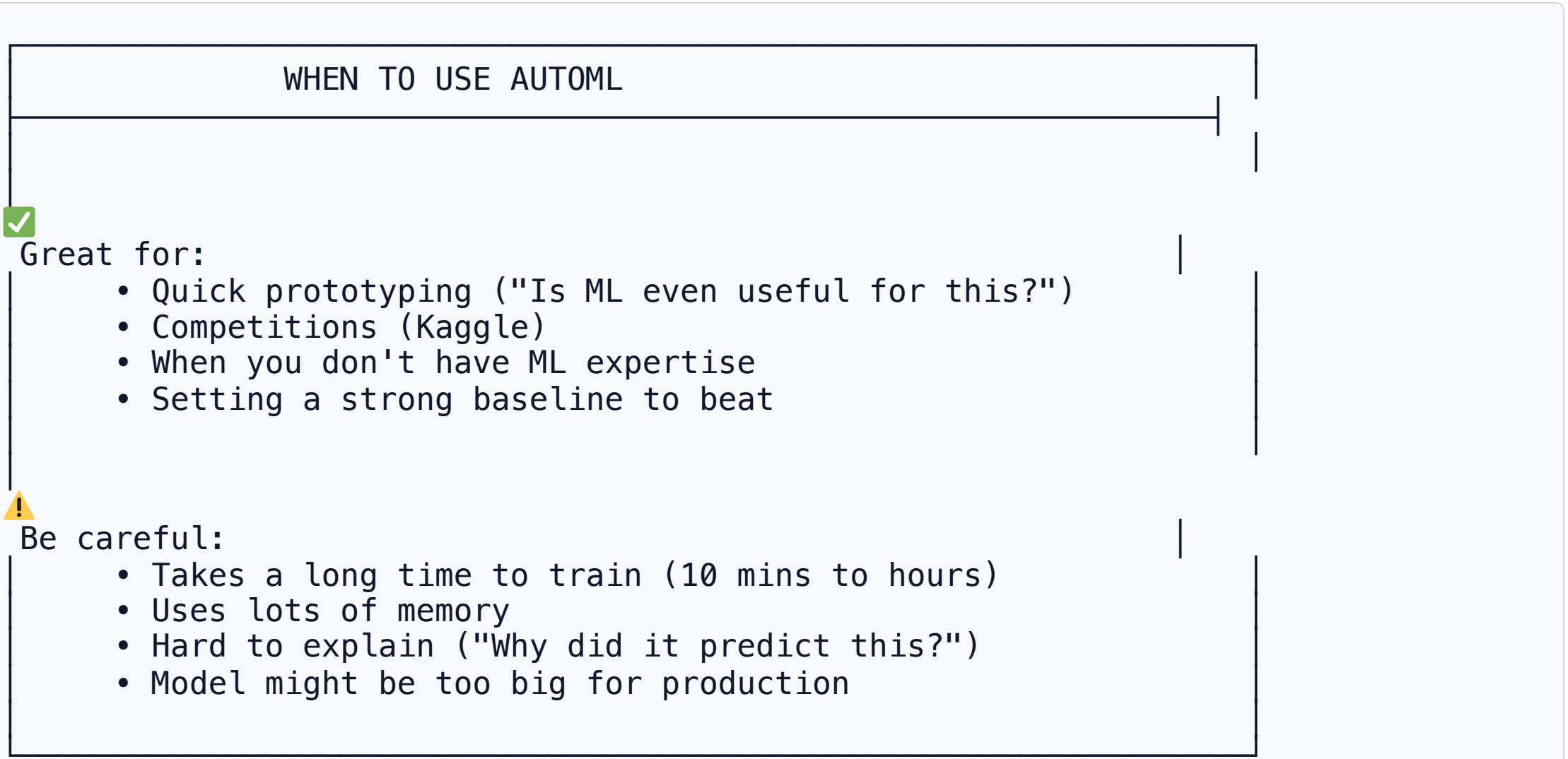
After training, you can see how each model performed:

```
predictor.leaderboard(test_data)
```

```
                 model  score_val  fit_time
0   WeightedEnsemble_L2       0.87       120s
1              CatBoost       0.85        45s
2              LightGBM       0.84        30s
3               XGBoost       0.83        50s
4          RandomForest       0.82        25s
5   NeuralNetFastAI     0.80        90s
6     LogisticRegression  0.76         5s
```

The ensemble combines the best models!

When to Use AutoML



AutoGluon with Time Limit

Don't have all day? Set a time limit:

```
predictor = TabularPredictor(label='success')

# Only train for 5 minutes
predictor.fit(train_data, time_limit=300)  # 300 seconds = 5 mins
```

More time = Better models (usually)

Time Limit	What AutoGluon Can Do
1 minute	Quick baselines (RF, LR)
5 minutes	Good models (+ XGBoost, LightGBM)
30 minutes	Great models (+ Neural Nets, tuning)
2+ hours	Best possible (full tuning, stacking)

Part 5: Transfer Learning

Standing on the shoulders of giants

The Problem with Training from Scratch

TRAINING A NEW MODEL

Scenario: You want to classify movie posters (images)

Option 1: Train from scratch
- Need: 1 million labeled images
- Need: 10 GPUs for a week
- Need: ML PhD to get it right
- Cost: $10,000+

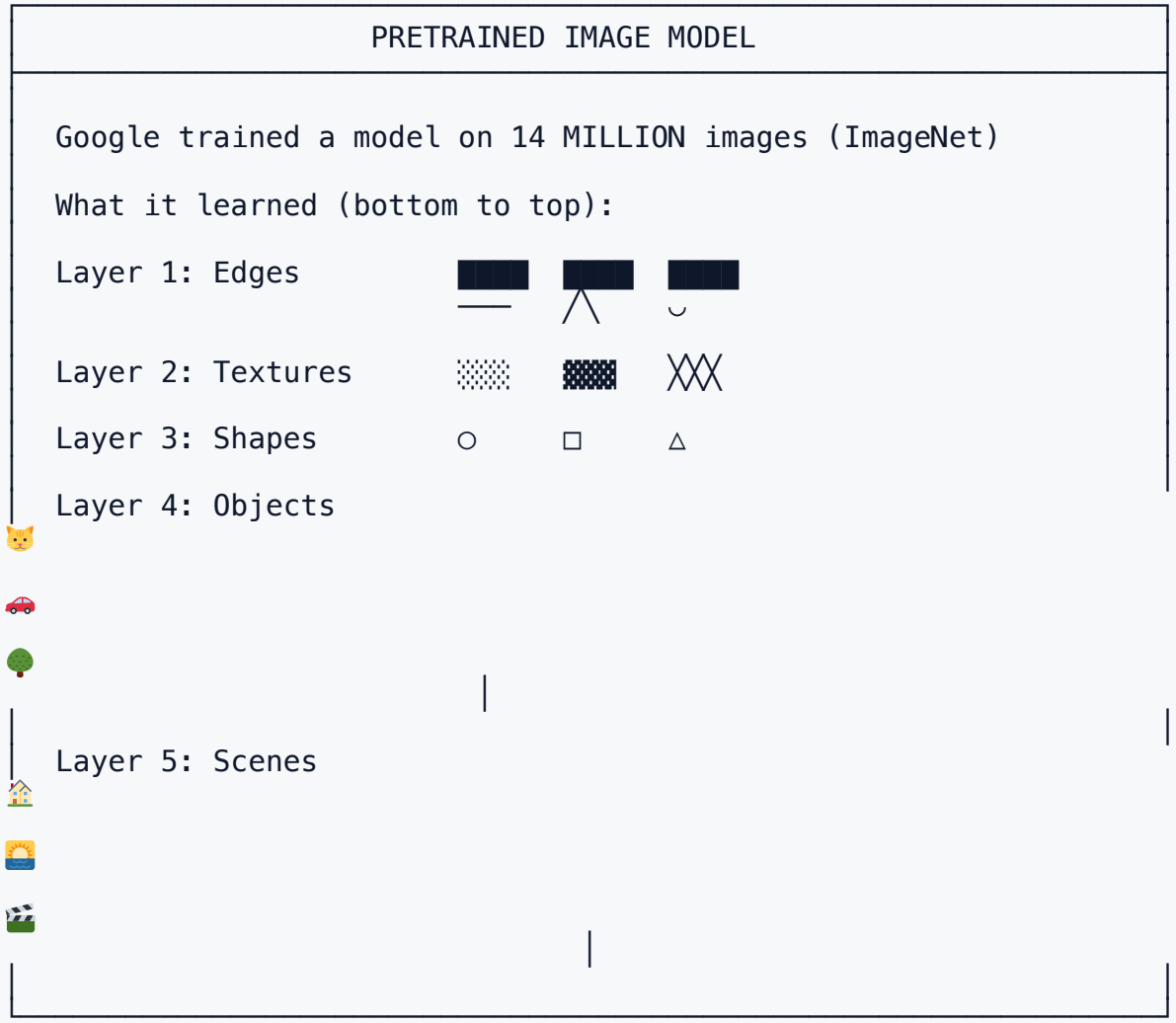
Option 2: Use someone else's model
- Need: 1,000 labeled images
- Need: 1 GPU for an hour
- Need: Basic Python skills
- Cost: $1

Transfer Learning: The Analogy

```
┌──────────────────────────────────────────────────────────────┐
│                  LEARNING TO PLAY A NEW SPORT                │
├──────────────────────────────────────────────────────────────┤
│                                                              │
│  Someone who has NEVER played any sport:                     │
│  • Learning tennis takes 6 months                            │
│  • Starts from zero                                          │
│                                                              │
│  Someone who plays badminton:                                │
│  • Learning tennis takes 2 months                            │
│  • Already knows: hand-eye coordination, racket grip,        │
│    court movement, strategy                                  │
│  • Just needs to learn: different swing, ball bounce         │
│                                                              │
│  The badminton player TRANSFERS their skills!                │
│                                                              │
└──────────────────────────────────────────────────────────────┘
```


Transfer Learning: Use knowledge from one task for another

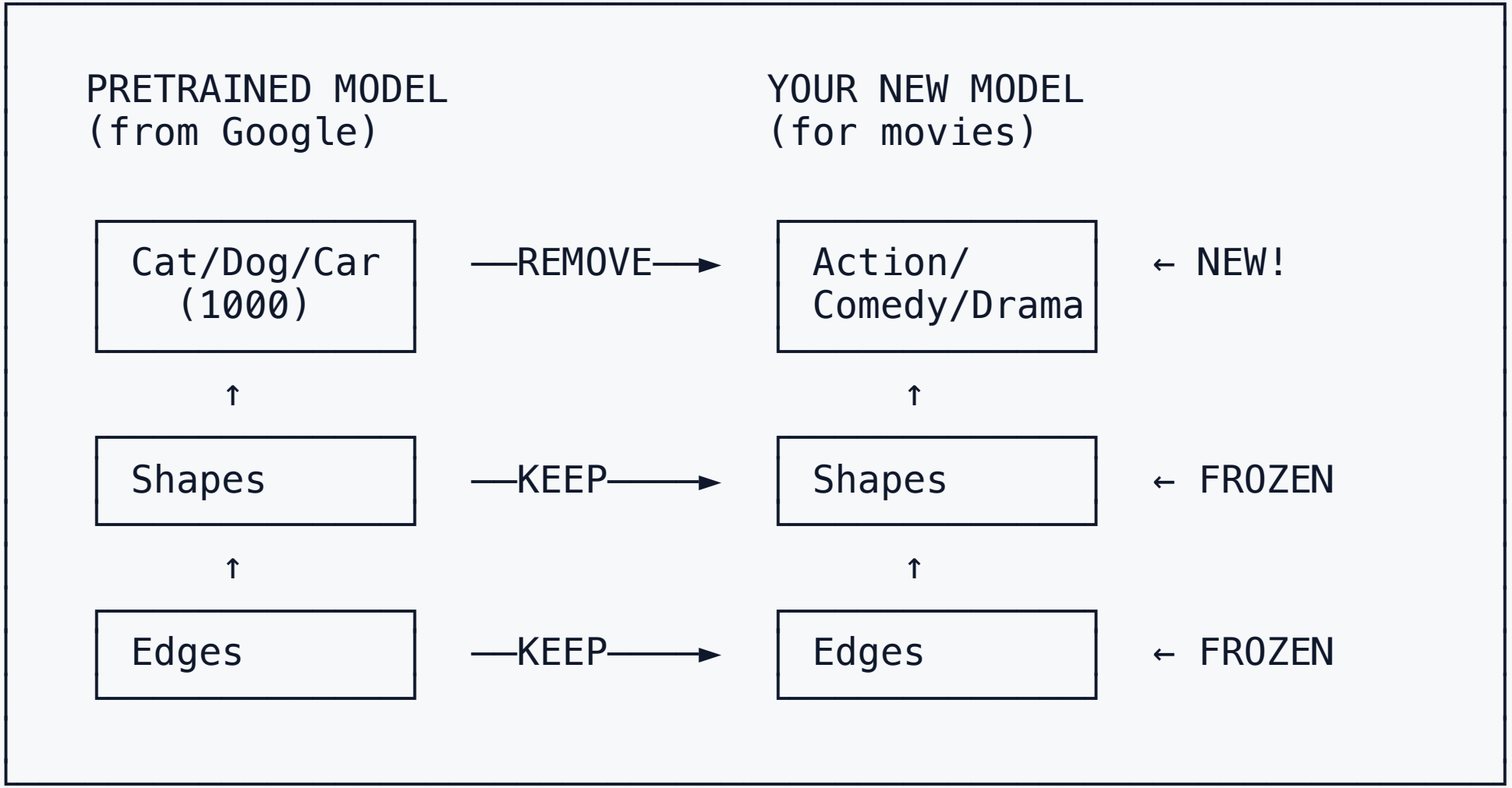
How Transfer Learning Works for Images



Transfer Learning Strategy

```
                    THE TRANSFER RECIPE

Step 1: Take a pretrained model (trained on millions of
        images by Google/Facebook)

Step 2: Remove the last layer (the "head")
        • Original: predicts 1000 ImageNet categories
        • We don't need "cat", "dog", "airplane"

Step 3: Add our own head
        • New layer: predicts OUR categories
        • Movie poster → "Action", "Comedy", "Drama"

Step 4: Train only the new head (freeze everything else)
        • Very fast! (minutes instead of days)
```

Transfer Learning Visualized



Transfer Learning for Text (LLMs)

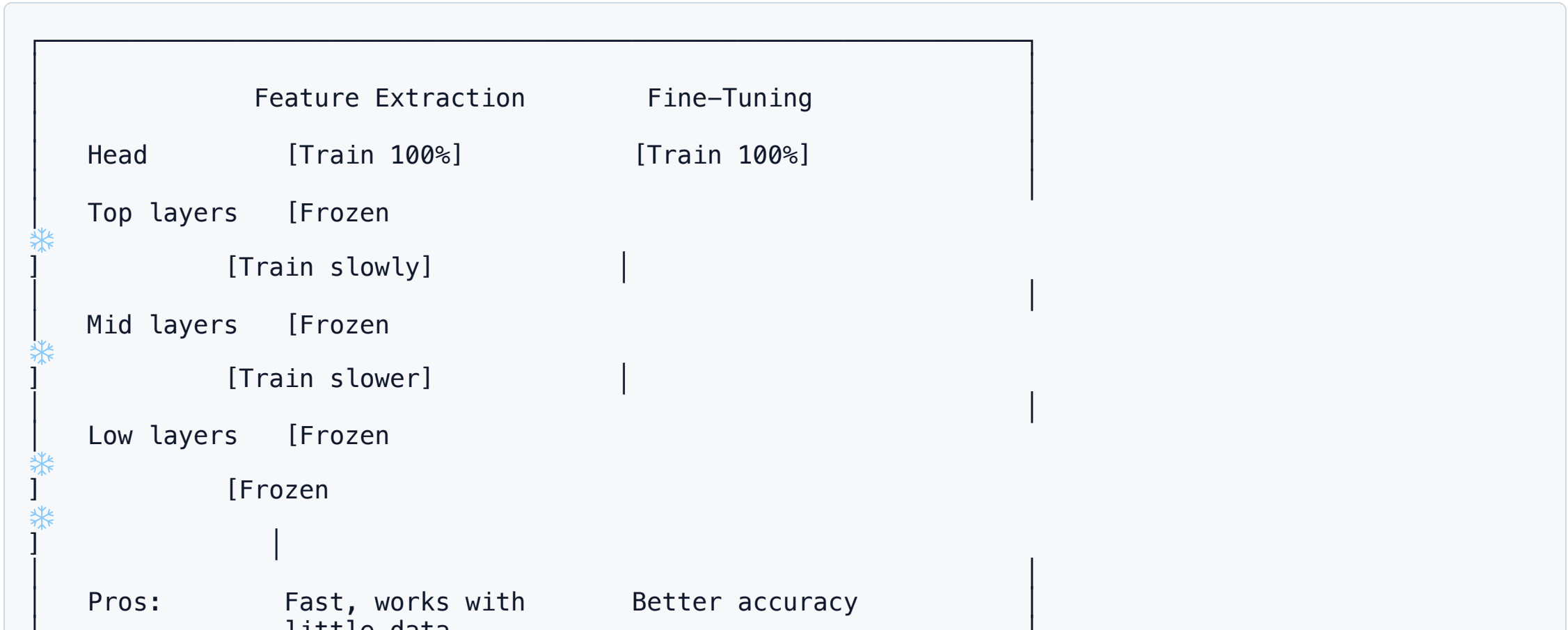
Same idea works for text!

```
┌──────────────────────────────────────────────────────────────┐
│                  PRETRAINED TEXT MODEL (BERT)                │
└──────────────────────────────────────────────────────────────┘

Google trained BERT on ALL of Wikipedia + Books

What it learned:
• Grammar and syntax
• Word meanings and relationships
• Common knowledge ("Paris is in France")
• Context understanding

Your task: Classify movie reviews as Positive/Negative

Transfer: Use BERT's language understanding,
          just teach it your specific task
```

Fine-Tuning: A Deeper Transfer

Feature Extraction: Freeze pretrained layers, only train new head

Fine-Tuning: Also slightly update the pretrained layers



When to Use Transfer Learning

```
TRANSFER LEARNING DECISION GUIDE

You have IMAGES?
→ Use pretrained ResNet, EfficientNet, or ViT
→ Works great even with 100 images!

You have TEXT?
→ Use pretrained BERT, RoBERTa, or use LLM APIs
→ Works great for classification, sentiment, etc.

You have TABULAR DATA (spreadsheets)?
→ Transfer learning is less common
→ Use AutoML instead (AutoGluon)

You have AUDIO?
→ Use pretrained Whisper, Wav2Vec
```

Transfer Learning Example Code

```
from transformers import pipeline

# Load a pretrained sentiment classifier
classifier = pipeline("sentiment-analysis")

# Use it immediately - no training needed!
reviews = [
    "This movie was absolutely fantastic!",
    "Worst film I've ever seen.",
    "It was okay, nothing special."
]

for review in reviews:
    result = classifier(review)
    print(f"{review[:30]}... → {result[0]['label']}")
```

Output:

Part 6: Putting It All Together

A complete workflow

The Complete Workflow

```
                    ML MODEL DEVELOPMENT WORKFLOW

Step 1: Understand your data
        • What type? (tabular, images, text)
        • How much? (100 samples vs 1 million)

Step 2: Create a baseline
        • Tabular: Logistic Regression or Random Forest
        • Images/Text: Pretrained model (transfer learning)

Step 3: Evaluate with cross-validation
        • Get reliable accuracy estimates
        • Understand variance in performance

Step 4: Try AutoML (if tabular)
        • Let AutoGluon find the best model
        • Compare to your baseline

Step 5: Iterate and improve
```

Netflix Movie Prediction: Full Example

```
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from autogluon.tabular import TabularPredictor

# Load our movie data
movies = pd.read_csv('movies.csv')

# Baseline: Random Forest with cross-validation
rf = RandomForestClassifier(n_estimators=100)
baseline_scores = cross_val_score(rf, X, y, cv=5)
print(f"Baseline (RF): {baseline_scores.mean():.1%} ± {baseline_scores.std():.1%}")

# AutoML: Let AutoGluon do its magic
predictor = TabularPredictor(label='success')
predictor.fit(movies, time_limit=300)
print(predictor.leaderboard())
```

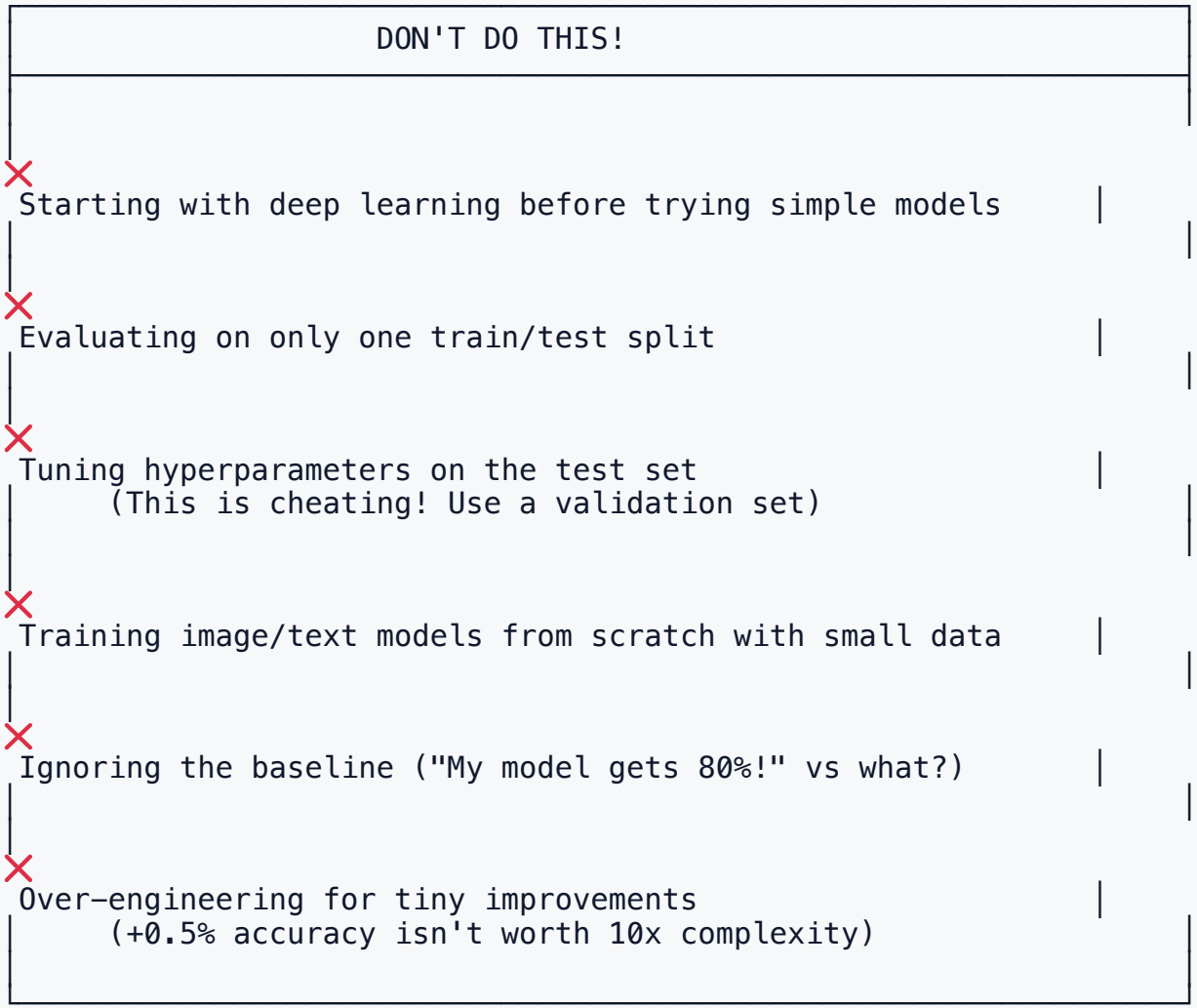
What Good Accuracy Looks Like

```
                INTERPRETING YOUR RESULTS

Random guessing:                50%
Majority class baseline:        60%
Simple model (Logistic Reg):    72%
Better model (Random Forest):   78%
AutoML (AutoGluon):             82%
State-of-the-art:               85%

Key questions:
• Did you beat random guessing? ✓
• Did you beat majority class? ✓
• Is the improvement worth the complexity?

82% might be amazing for some problems,
and terrible for others. Context matters!
```

Key Takeaways

```
                    TODAY'S KEY LESSONS

1. ALWAYS START WITH A BASELINE
   → Simple models are your reference point
   → You can't know if fancy is better without simple first

2. USE CROSS-VALIDATION
   → One test set can be misleading
   → 5-fold CV gives reliable estimates

3. TRY AUTOML FOR TABULAR DATA
   → AutoGluon does the hard work for you
   → Great for prototyping and competitions

4. USE TRANSFER LEARNING FOR IMAGES/TEXT
   → Don't train from scratch
   → Pretrained models save time and work better
```

Common Mistakes to Avoid



Next Week Preview

```
┌──────────────────────────────────────────────────────────────────────┐
│                        COMING UP: WEEK 8                             │
└──────────────────────────────────────────────────────────────────────┘

Model Evaluation & Deployment

• Confusion matrices (understanding errors)
• Precision, Recall, F1 (beyond accuracy)
• When accuracy is misleading
• Deploying your model to production

You've built the model. Now how do you know it's REALLY good?
```

Lab Preview

This week's hands-on exercises:

1. **Build baselines**: Compare Logistic Regression, Decision Tree, Random Forest
2. **Cross-validate**: Use 5-fold CV to get reliable estimates
3. **Try AutoGluon**: Let it find the best model for Netflix data
4. **Transfer learning demo**: Use a pretrained model for text classification

All code will be provided. Focus on understanding!

Questions?

Key concepts:

- Baseline models
- Cross-validation
- AutoML (AutoGluon)
- Transfer learning

Remember: Simple first, complex only if needed!