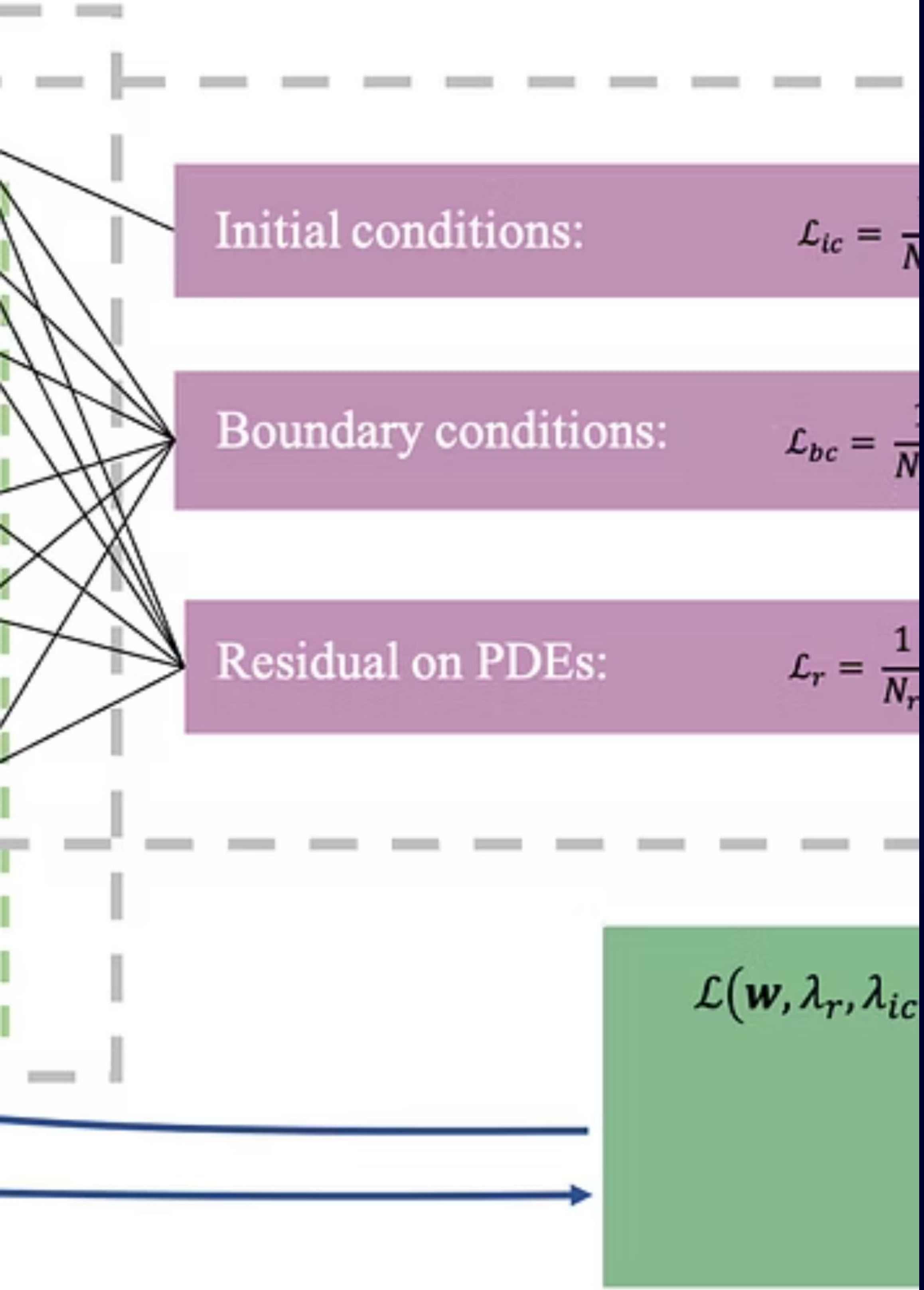


# Introduction to Machine Learning

Hernán Andrés Morales-Navarrete

# Today's Agenda

01

## Foundations of ML

What is machine learning and why is it transforming physics and computer science research today?

02

## ML Categories

Supervised learning, unsupervised learning, and a glimpse into reinforcement learning

03

## Training & Generalization

Understanding how models learn from data and perform on unseen examples

04

## Cross-Validation

Practical techniques for robust model evaluation and selection

05

## Overfitting Demonstration

Visual examples and code showing when models memorize instead of learn

# What is Machine Learning?

Machine learning represents a fundamental shift in how we approach problem-solving: instead of explicitly programming rules, we learn patterns directly from data. The paradigm moves from hand-coded logic to data-driven discovery.

## Traditional Approach

Data + Rules → Answers

- Expert defines explicit logic
- Limited to known patterns
- Struggles with complexity

## Machine Learning

Data + Targets → Model (Rules)

- Algorithm discovers patterns
- Handles high dimensions
- Adapts to complexity

## Physics Intuition

Think of ML as **generalized curve fitting with sophisticated regularization**. While traditional physics might involve fitting spectral lines to known functions, machine learning can discover complex mappings, from raw images or field configurations to physical properties, without assuming functional forms.

### Experiments

Detector readings, sensor arrays

### Simulations

Computational physics outputs

### Observations

Telescopes, microscopes

# Types of Machine Learning

Machine learning encompasses several distinct paradigms, each suited to different types of problems. Today we'll focus primarily on supervised and unsupervised approaches, with a brief look at reinforcement learning for completeness.

## Supervised Learning

Learn a function $f: X \to Y$ from labeled training examples. The algorithm is "supervised" by providing correct answers during training.

- **Regression:** Predict continuous values (energy levels, temperatures, cross-sections)
- **Classification:** Assign discrete labels (particle types, phases, defect detection)

## Unsupervised Learning

Discover hidden structure in data without labeled examples. The algorithm finds patterns autonomously.

- **Clustering:** Group similar data points (regime discovery, pattern recognition)
- **Dimensionality reduction:** Compress high-dimensional data while preserving structure

## Reinforcement Learning

Agent learns optimal behavior through trial-and-error interaction with an environment, guided by reward signals.

- Experiment planning and adaptive control
- Hyperparameter optimization strategies

# Supervised Learning: Regression

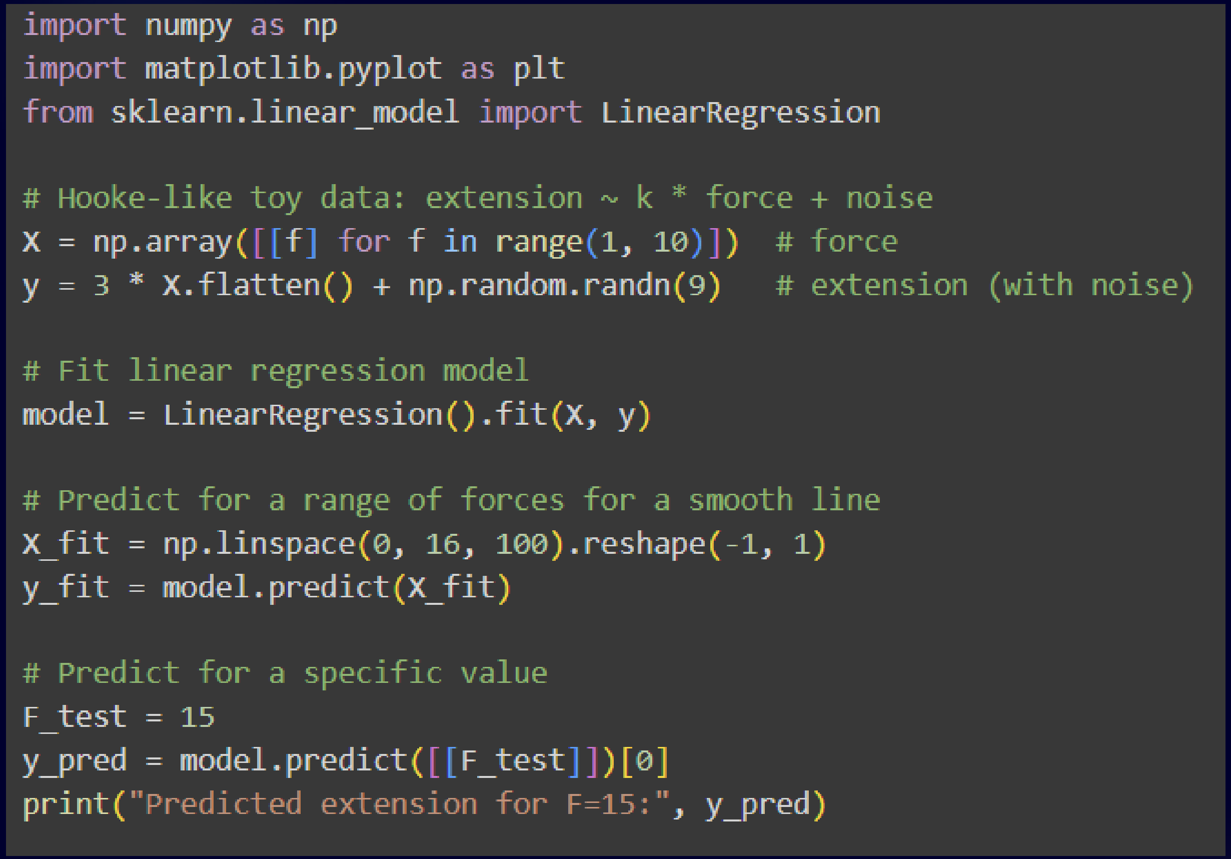
In regression tasks, we predict **continuous numerical values** from input features.

This is perhaps the most intuitive ML paradigm for physicists, as it generalizes familiar curve-fitting procedures.

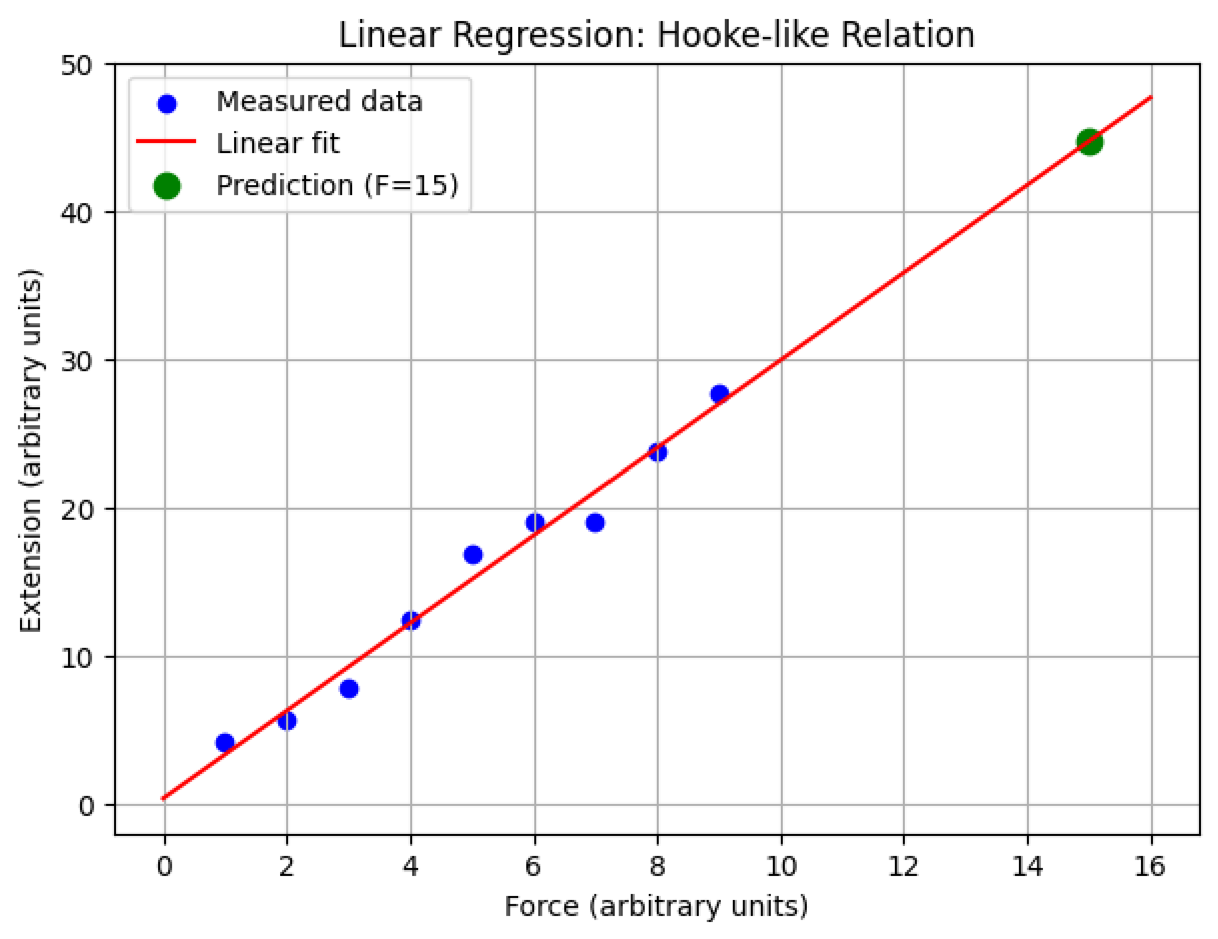
## Examples

- Predict material properties (conductivity, hardness) from chemical composition
- Map simulation parameters to observable quantities
- Estimate particle energies from detector signals
- Interpolate potential energy surfaces

Let's explore a physics-inspired example using Hooke's law as our toy model. We'll learn the spring constant from noisy experimental data.



```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Hooke-like toy data: extension ~ k * force + noise
X = np.array([[f] for f in range(1, 10)])  # force
y = 3 * X.flatten() + np.random.randn(9)   # extension (with noise)

# Fit linear regression model
model = LinearRegression().fit(X, y)

# Predict for a range of forces for a smooth line
X_fit = np.linspace(0, 16, 100).reshape(-1, 1)
y_fit = model.predict(X_fit)

# Predict for a specific value
F_test = 15
y_pred = model.predict([[F_test]])[0]
print("Predicted extension for F=15:", y_pred)
```



**Physical interpretation:** The learned coefficients represent physical proportionality constants (like spring constants) extracted from noisy measurements. The model captures the underlying relationship despite experimental uncertainty.

# Supervised Learning: Classification

Classification assigns discrete labels to inputs. In physics, this appears whenever we need to categorize states, identify particles, or detect phases.

### Particle Identification

High-energy physics experiments use classification to distinguish electrons, muons, pions, and other particles based on detector signatures. Neural networks now achieve near-human expert performance.

### Phase Classification

Identify phases of matter (ferromagnetic, paramagnetic, superconducting) from simulation snapshots or experimental measurements. ML can discover phase boundaries without prior knowledge.

### Defect Detection

Automated identification of defects, anomalies, or rare events in microscopy images, material samples, or experimental data streams.

## Common Classification Models

- **Logistic Regression:** Linear decision boundaries, interpretable
- **Support Vector Machines (SVM):** Optimal margin separation, kernel tricks
- **Random Forest:** Ensemble of decision trees, robust to noise
- **Gradient Boosting:** Sequential improvement of weak learners
- **Neural Networks:** Flexible, learns hierarchical features (deep dive in next class)

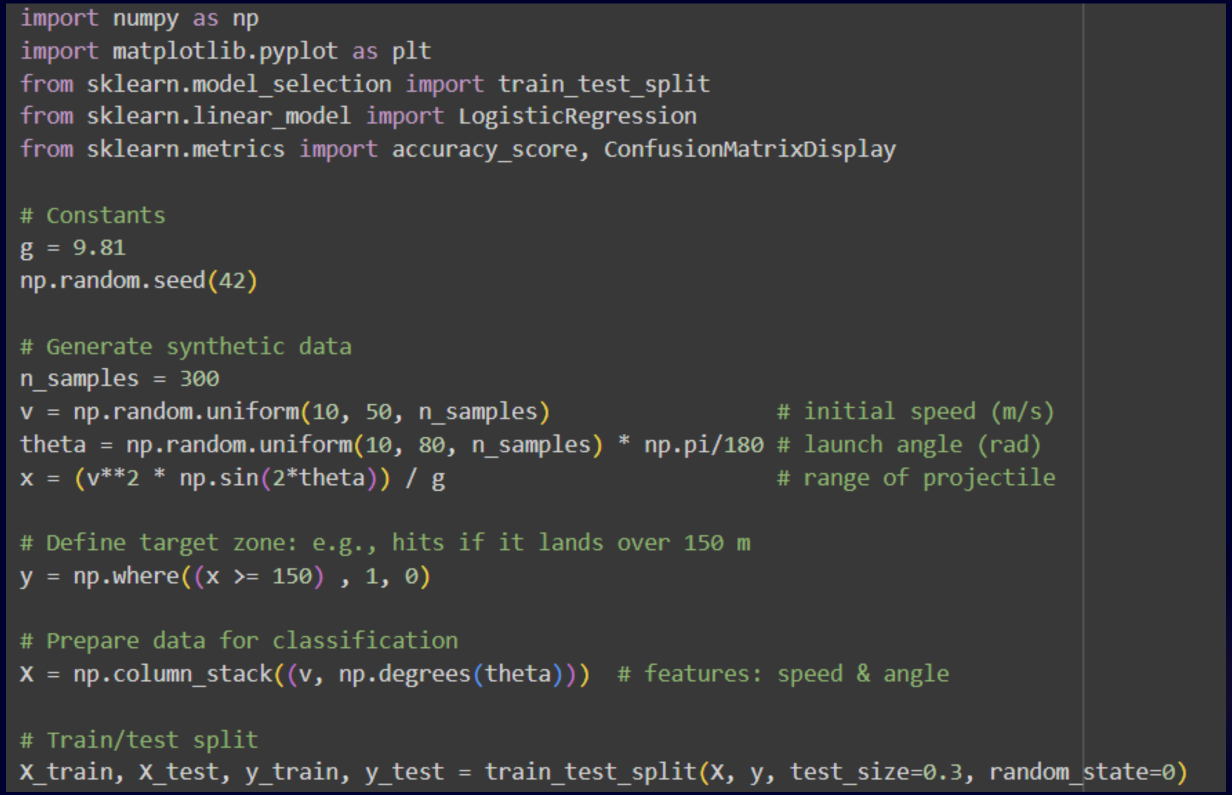
# Supervised Learning: Classification

We simulate many projectiles using basic physics:

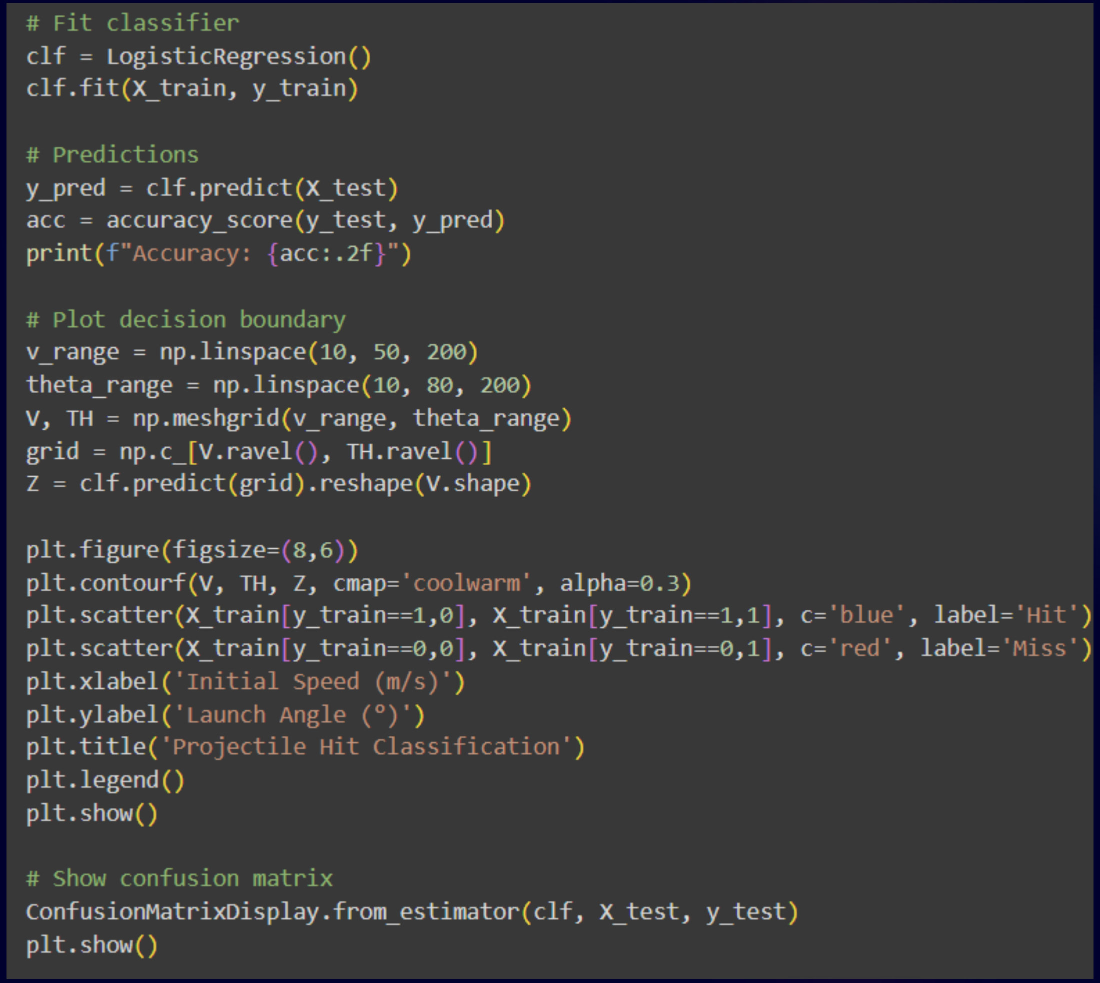
$$x = \frac{v^2 \sin(2\theta)}{g}$$

and label them as
•1 (hit target) if they land near a target distance,
•0 (miss) otherwise.

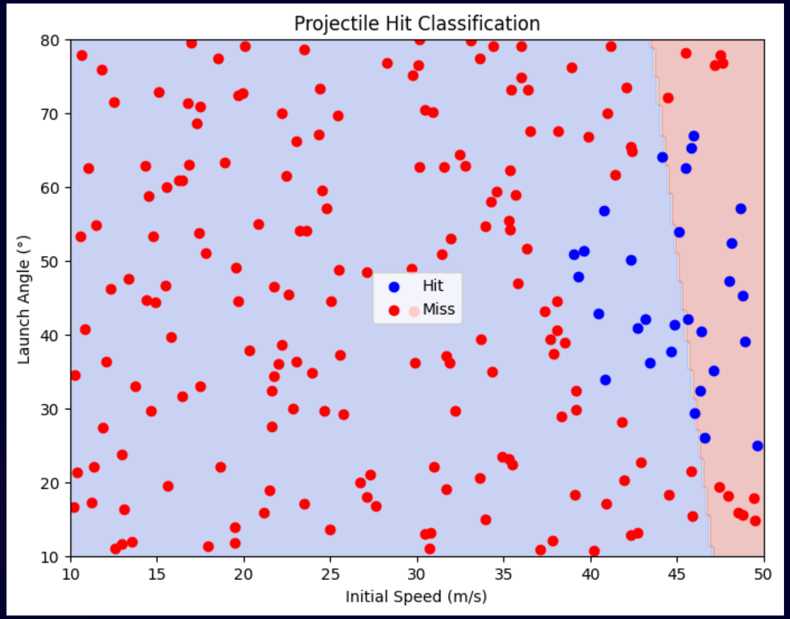
Then we train a **Logistic Regression** classifier to learn the decision boundary.

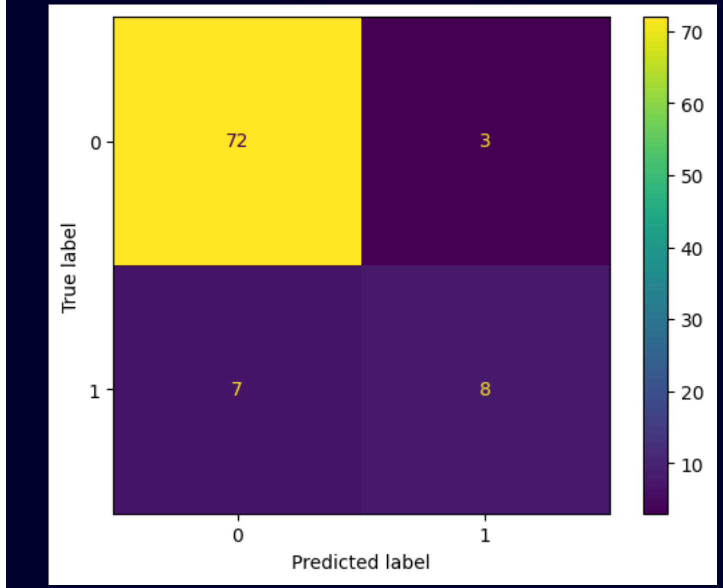


```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, ConfusionMatrixDisplay

# Constants
g = 9.81
np.random.seed(42)

# Generate synthetic data
n_samples = 300
v = np.random.uniform(10, 50, n_samples)                   # initial speed (m/s)
theta = np.random.uniform(10, 80, n_samples) * np.pi/180 # launch angle (rad)
x = (v**2 * np.sin(2*theta)) / g                         # range of projectile

# Define target zone: e.g., hits if it lands over 150 m
y = np.where((x >= 150) , 1, 0)

# Prepare data for classification
X = np.column_stack((v, np.degrees(theta)))  # features: speed & angle

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```



```
# Fit classifier
clf = LogisticRegression()
clf.fit(X_train, y_train)

# Predictions
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {acc:.2f}")

# Plot decision boundary
v_range = np.linspace(10, 50, 200)
theta_range = np.linspace(10, 80, 200)
V, TH = np.meshgrid(v_range, theta_range)
grid = np.c_[V.ravel(), TH.ravel()]
Z = clf.predict(grid).reshape(V.shape)

plt.figure(figsize=(8,6))
plt.contourf(V, TH, Z, cmap='coolwarm', alpha=0.3)
plt.scatter(X_train[y_train==1,0], X_train[y_train==1,1], c='blue', label='Hit')
plt.scatter(X_train[y_train==0,0], X_train[y_train==0,1], c='red', label='Miss')
plt.xlabel('Initial Speed (m/s)')
plt.ylabel('Launch Angle (°)')
plt.title('Projectile Hit Classification')
plt.legend()
plt.show()

# Show confusion matrix
ConfusionMatrixDisplay.from_estimator(clf, X_test, y_test)
plt.show()
```





# Unsupervised Learning: Clustering

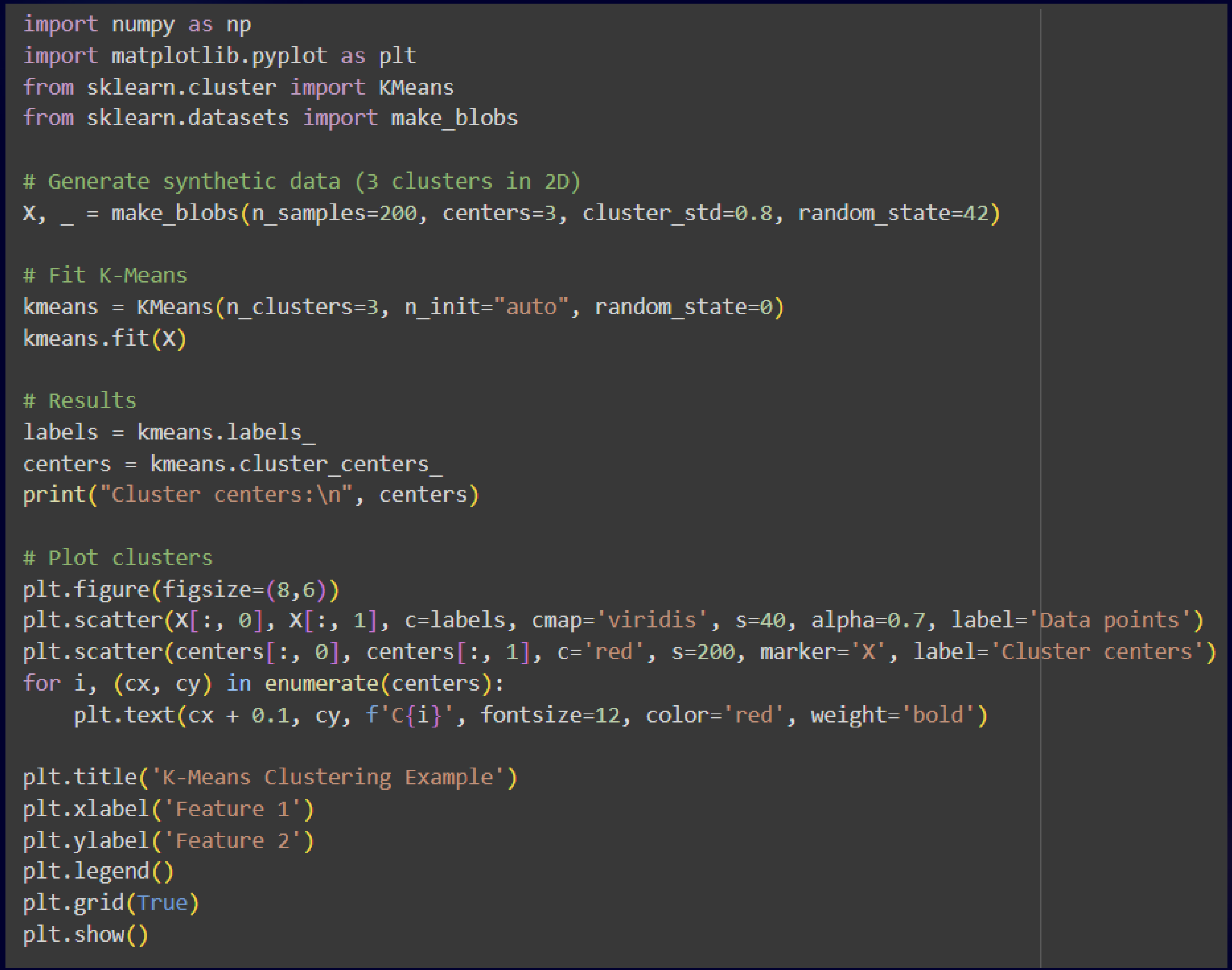
Clustering algorithms discover natural groupings in data **without predefined labels**. This is invaluable when exploring new parameter spaces or seeking regime boundaries.

## Physics Applications

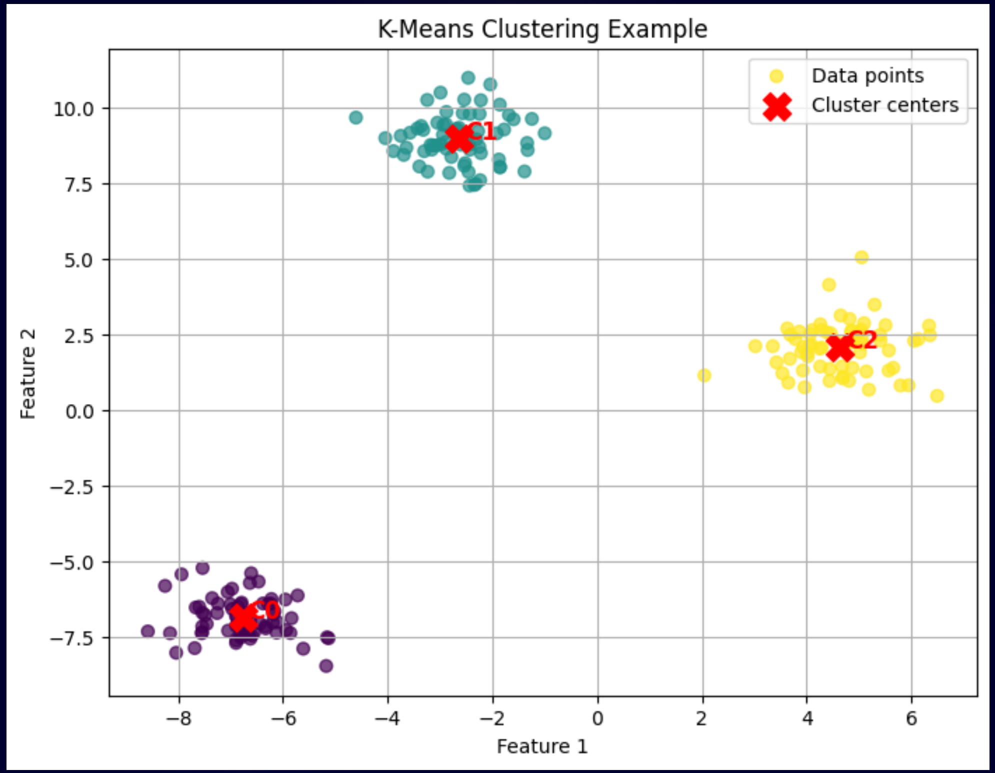
- **Spectral grouping:** Classify astronomical objects by spectral features
- **Trajectory analysis:** Identify distinct behavioral regimes in molecular dynamics
- **State discovery:** Find ordered vs. disordered regimes in time-series data
- **Configuration space:** Partition parameter space into physically meaningful regions

## Mini-Demo: K-Means Clustering

K-means partitions data into *k* clusters by iteratively assigning points to the nearest cluster center and updating centers.



```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate synthetic data (3 clusters in 2D)
X, _ = make_blobs(n_samples=200, centers=3, cluster_std=0.8, random_state=42)

# Fit K-Means
kmeans = KMeans(n_clusters=3, n_init="auto", random_state=0)
kmeans.fit(X)

# Results
labels = kmeans.labels_
centers = kmeans.cluster_centers_
print("Cluster centers:\n", centers)

# Plot clusters
plt.figure(figsize=(8,6))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=40, alpha=0.7, label='Data points')
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, marker='X', label='Cluster centers')
for i, (cx, cy) in enumerate(centers):
    plt.text(cx + 0.1, cy, f'C{i}', fontsize=12, color='red', weight='bold')

plt.title('K-Means Clustering Example')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid(True)
plt.show()
```



**Choosing k:** Use the elbow method (plot within-cluster variance vs. k) or silhouette scores to determine the optimal number of clusters. Domain knowledge can also guide this choice.

# Unsupervised Learning: Dimensionality Reduction

Real-world physics data often lives in extremely high-dimensional spaces—think of each pixel in an image or each time point in a simulation as a separate dimension. Dimensionality reduction techniques compress this complexity while preserving essential structure.

### PCA / SVD

**Principal Component Analysis** finds linear combinations of features that capture maximum variance. Perfect for understanding dominant modes or noise reduction through truncation.

### t-SNE / UMAP

**Nonlinear embeddings** preserve local neighborhood structure, revealing clusters and manifolds invisible to linear methods. Excellent for visualization and exploratory analysis.

### Autoencoders

Neural networks that **compress and reconstruct** data, learning nonlinear latent representations. We'll explore these in detail in an upcoming class on deep learning.

## Why This Matters for Physics

### Visualization

Project complex phase spaces into 2D/3D for human interpretation and regime identification

### Noise Reduction

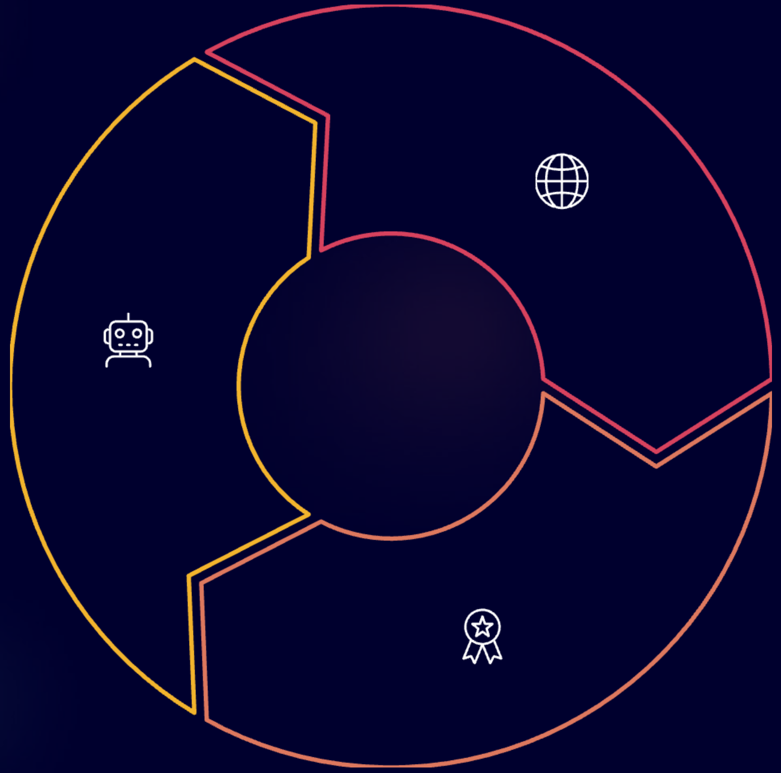
Separate signal from noise by projecting onto principal components

### Efficiency

Compress field configurations for faster downstream analysis and storage

# Reinforcement Learning: A Glimpse

While supervised and unsupervised learning dominate current physics applications, **reinforcement learning (RL)** offers exciting possibilities for optimization and control problems.



Agent

Takes actions based on current state

Environment

Responds to actions, provides new state

Reward

Feedback signal guiding learning

## Physics Applications

- **Experiment planning:** Adaptively choose next measurements to maximize information gain
- **Control systems:** Optimize tokamak plasma control or quantum gate sequences
- **Hyperparameter search:** Efficiently explore model configuration spaces

RL is computationally intensive and requires careful reward engineering. We'll revisit this in depth during later sessions on optimization and active learning strategies.

# Training, Validation, and Test Sets

Proper data splitting is **absolutely critical** for honest model evaluation. Violating these principles leads to overly optimistic performance estimates and models that fail in practice.

## Training Set

**Purpose:** Fit model parameters (weights, coefficients)

**Typical size:** 60–80% of data

The model sees and learns from these examples repeatedly during optimization.

## Validation Set

**Purpose:** Tune hyperparameters and select among model architectures

**Typical size:** 10–20% of data

Used to compare models and make development decisions, but not for final reporting.

## Test Set

**Purpose:** Provide unbiased estimate of generalization performance

**Typical size:** 10–20% of data

Touched only once, after all development is complete. This is your ground truth.

---

⚠ **Data leakage =** Any information from validation or test sets that influences model development compromises your evaluation. Keep test data completely isolated until final assessment. Even looking at test set properties (mean, variance) can introduce subtle biases.

# Cross-Validation in Practice

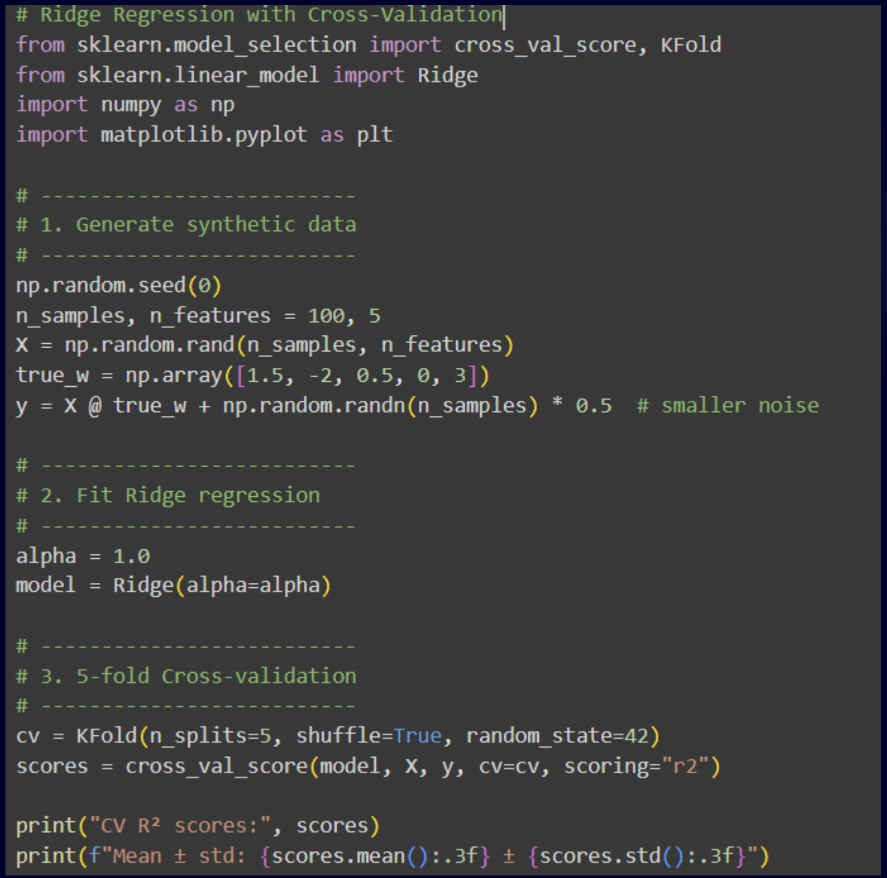
**Cross-validation (CV)** provides a more robust estimate of model performance by averaging across multiple train/validation splits. This is especially valuable when working with limited datasets, a common scenario in experimental physics.

## How K-Fold CV Works

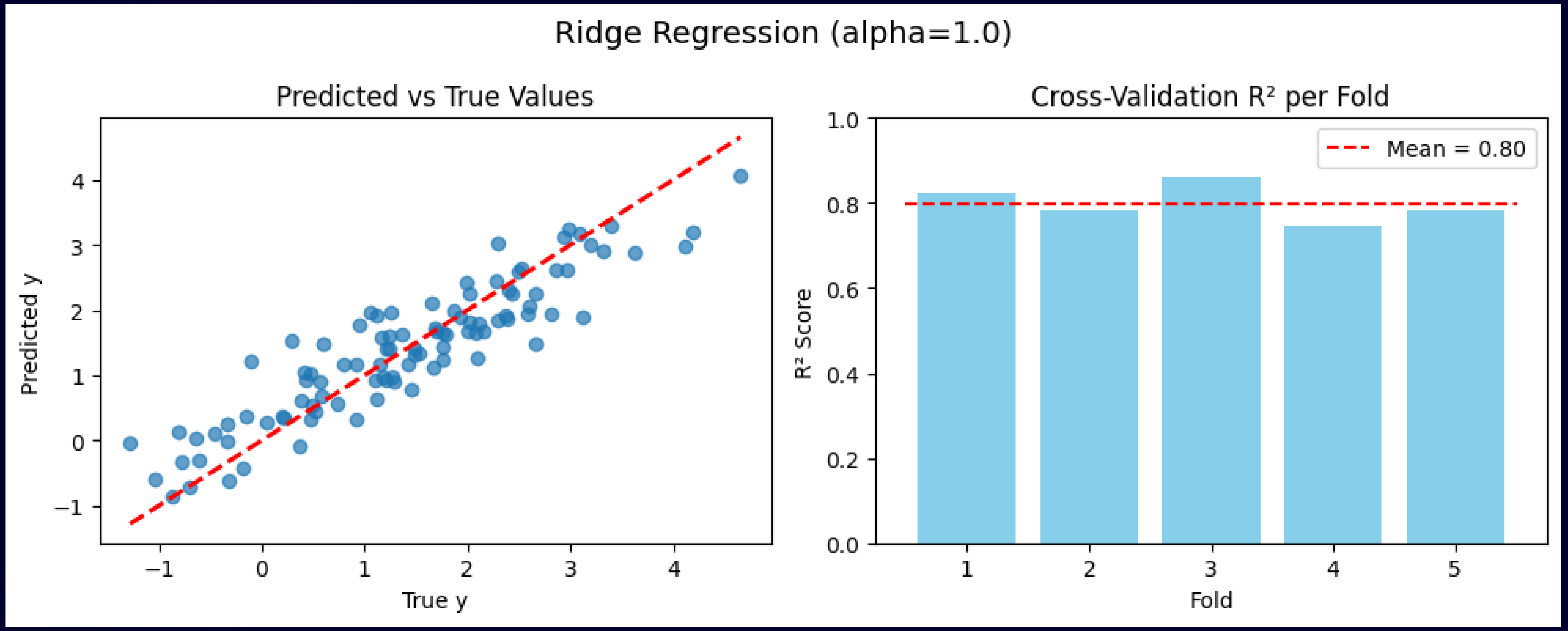
1. Partition data into *k* equal-sized folds
2. For each fold: train on *k*-1 folds, validate on the held-out fold
3. Average the *k* validation scores to get overall performance estimate
4. Standard deviation across folds indicates stability

## Mini-Demo: Ridge Regression with 5-Fold CV

We'll generate synthetic data with known structure and use cross-validation to assess how well Ridge regression recovers it.



```
# Ridge Regression with Cross-Validation
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import Ridge
import numpy as np
import matplotlib.pyplot as plt

# -------------------------
# 1. Generate synthetic data
# -------------------------
np.random.seed(0)
n_samples, n_features = 100, 5
X = np.random.rand(n_samples, n_features)
true_w = np.array([1.5, -2, 0.5, 0, 3])
y = X @ true_w + np.random.randn(n_samples) * 0.5  # smaller noise

# -------------------------
# 2. Fit Ridge regression
# -------------------------
alpha = 1.0
model = Ridge(alpha=alpha)

# -------------------------
# 3. 5-fold Cross-validation
# -------------------------
cv = KFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(model, X, y, cv=cv, scoring="r2")

print("CV R² scores:", scores)
print(f"Mean ± std: {scores.mean():.3f} ± {scores.std():.3f}")
```



The **$R^2$ score** measures the proportion of variance explained by the model (1.0 = perfect, 0.0 = no better than mean).

**Why regularization?** Ridge regression (L2 penalty) prevents overfitting by penalizing large coefficients. The alpha parameter controls this tradeoff—tune it using CV!

# The Bias–Variance Tradeoff

Understanding the bias–variance tradeoff is fundamental to building models that generalize well. It explains why both too-simple and too-complex models fail.

## High Bias (Underfitting)

**Symptoms:**

- Poor performance on training data
- Model too simple for the problem
- Misses important patterns

**Example:** Fitting quadratic data with a line

**Solution:** Increase model capacity (more features, higher polynomial degree, deeper network)

## High Variance (Overfitting)

**Symptoms:**

- Excellent training, poor validation performance
- Model too flexible for available data
- Memorizes noise as signal

**Example:** Degree-10 polynomial through 15 noisy points

**Solution:** Reduce capacity, add regularization, collect more data

## The Sweet Spot

The goal is finding the **Goldilocks zone**: a model complex enough to capture true patterns but constrained enough to avoid memorizing noise. Cross-validation helps identify this balance by revealing when validation error starts increasing despite decreasing training error.

↓↓

### Training Error

Always decreases with model complexity

U

### Validation Error

Decreases then increases—the U-curve reveals optimal complexity

★

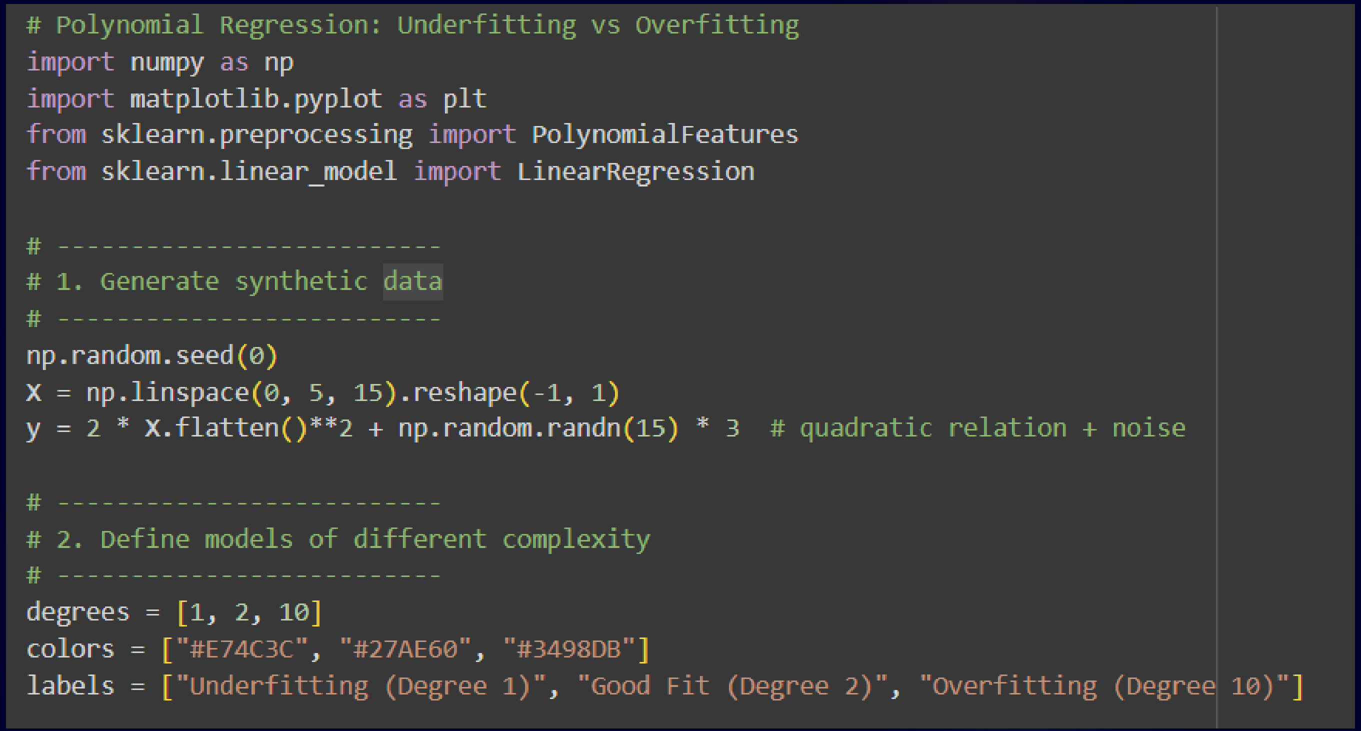
### Optimal Point

Minimize validation error, not training error

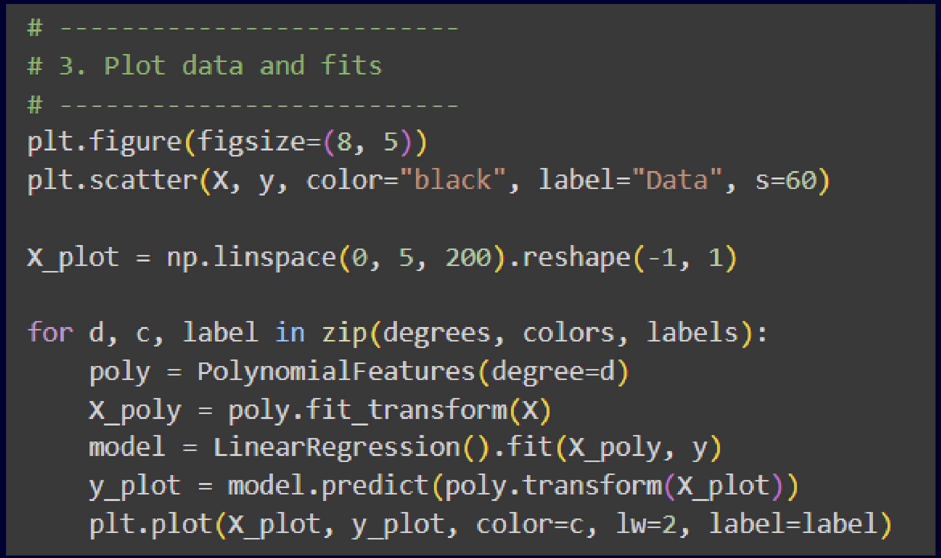
# Overfitting: Visual Demonstration

Let's make overfitting concrete with a polynomial regression example. We'll fit the same noisy experimental data with polynomials of increasing degree and observe the transition from underfitting to overfitting.

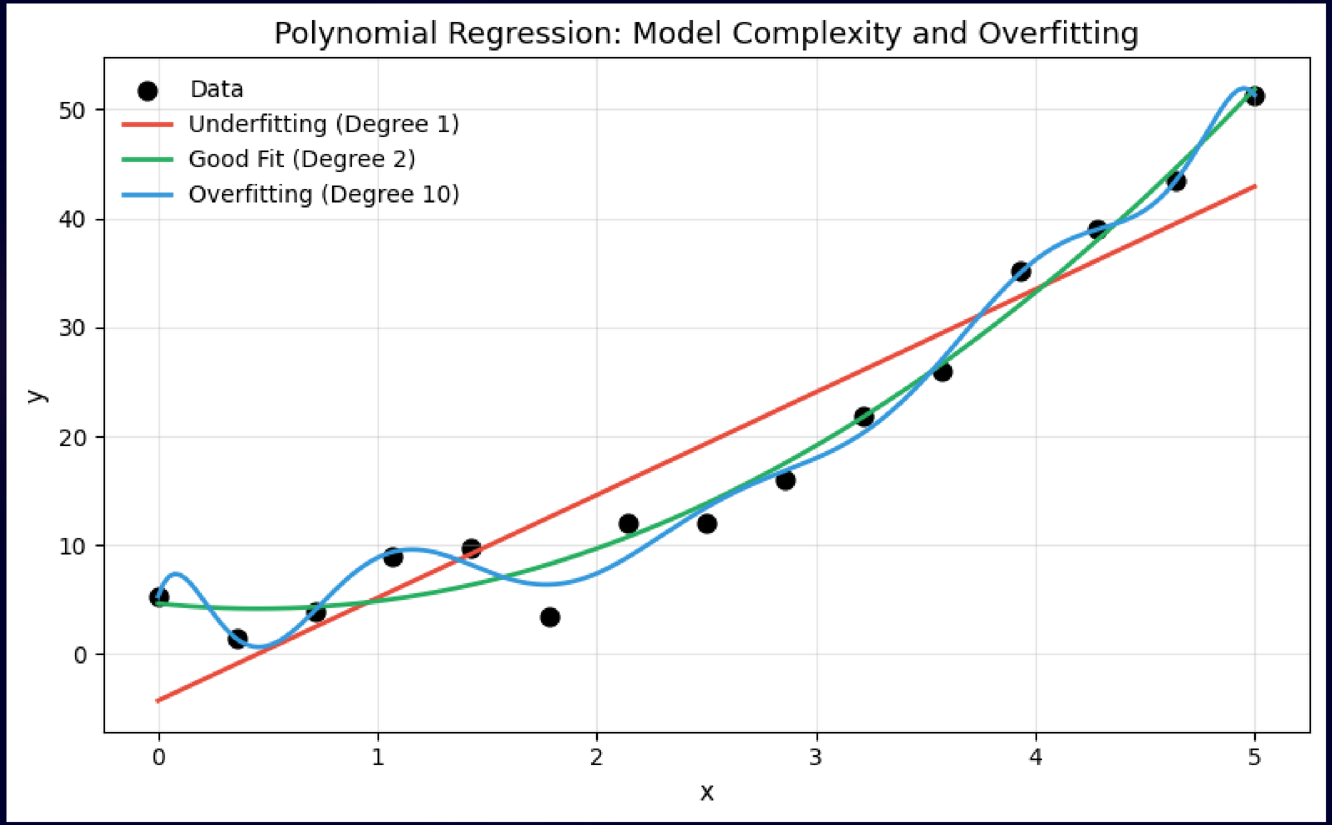
Imagine measuring a physical quantity that varies quadratically with an input parameter, but your measurements include experimental noise. How do different model complexities handle this?



```python
# Polynomial Regression: Underfitting vs Overfitting
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# -------------------------
# 1. Generate synthetic data
# -------------------------
np.random.seed(0)
X = np.linspace(0, 5, 15).reshape(-1, 1)
y = 2 * X.flatten()**2 + np.random.randn(15) * 3  # quadratic relation + noise

# -------------------------
# 2. Define models of different complexity
# -------------------------
degrees = [1, 2, 10]
colors = ["#E74C3C", "#27AE60", "#3498DB"]
labels = ["Underfitting (Degree 1)", "Good Fit (Degree 2)", "Overfitting (Degree 10)"]
```



```python
# -------------------------
# 3. Plot data and fits
# -------------------------
plt.figure(figsize=(8, 5))
plt.scatter(X, y, color="black", label="Data", s=60)

X_plot = np.linspace(0, 5, 200).reshape(-1, 1)

for d, c, label in zip(degrees, colors, labels):
    poly = PolynomialFeatures(degree=d)
    X_poly = poly.fit_transform(X)
    model = LinearRegression().fit(X_poly, y)
    y_plot = model.predict(poly.transform(X_plot))
    plt.plot(X_plot, y_plot, color=c, lw=2, label=label)
```



Degree 1 (Linear): Underfit
The straight line is too simple to capture the quadratic relationship.
High training error, but at least it doesn't wildly extrapolate.

Degree 2 (Quadratic): Just Right
Matches the true underlying relationship.
Balances fit quality with generalization. This is our target!

Degree 10: Overfit
Passes through nearly every noisy point but oscillates wildly between them.
Perfect training fit, terrible generalization. The model mistakes noise for signal.

# Regularization & Model Selection

Regularization techniques constrain model complexity to prevent overfitting. Think of them as encoding prior knowledge or preferences about solution properties.

1

### Ridge Regression (L2)

Penalizes large coefficient magnitudes: $\text{Loss} = \text{MSE} + \alpha \sum w_i^2$

Shrinks all coefficients toward zero proportionally. Prefers smooth, stable solutions. Good when many features contribute.

2

### Lasso Regression (L1)

Penalizes absolute coefficient values: $\text{Loss} = \text{MSE} + \alpha \sum |w_i|$

Drives some coefficients exactly to zero, performing automatic feature selection. Prefers sparse solutions.

3

### Early Stopping

For iterative algorithms (gradient descent, neural networks), stop training when validation error stops decreasing.

Prevents the model from continuing to memorize training noise.

4

### Dropout (Deep Learning)

Randomly deactivate neurons during training to prevent co-adaptation.

Forces network to learn robust, distributed representations. We'll explore this in the neural networks session.

## Model Capacity & Data Size

Choose your hypothesis class (model family and complexity) appropriate to your dataset size. A general principle: **Model capacity should scale with data quantity**.

- Small data (dozens to hundreds): Linear models, low-degree polynomials
- Medium data (thousands): Random forests, SVMs, shallow neural networks
- Large data (millions+): Deep neural networks, high-capacity models

**Physics priors:** Incorporate domain knowledge through regularization—enforce smoothness, symmetries, conservation laws, or known functional forms.

**Hyperparameter tuning:** Always tune regularization strength (α), model capacity, and other hyperparameters using cross-validation on training+validation sets. Never tune on test data!

# Practical ML Workflow

Success in applied machine learning requires a systematic approach. Here's a battle-tested workflow that prevents common pitfalls and ensures reproducible results.

Define the Problem
What physics question are you answering? What metric matters? (accuracy, precision, recall, RMSE, physical interpretability)

Split Your Data
Before any exploration: create train/validation/test splits. Set test aside and don't touch it until the very end.

Establish Baselines
Start simple! Linear regression, PCA, mean prediction. ML only adds value if it beats strong baselines.

Feature Engineering
Transform raw data: scaling, normalization, domain-specific features. Physics insight shines here: incorporate symmetries, conservation laws.

Train & Cross-Validate
Fit models with various hyperparameters. Use CV to compare objectively. Track what you've tried to avoid redundant experiments.

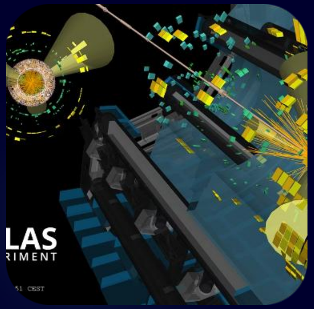
Error Analysis
Inspect failures. Where does the model struggle? Are errors random or systematic? Use physics intuition to diagnose issues.

Iterate & Document
Refine based on error analysis. Document decisions, hyperparameters, random seeds. Ensure others (including future you) can reproduce results.

**Golden rule:** Always establish strong baselines first. A simple linear model that captures 90% of the variance shows you where complexity actually adds value. Don't reach for neural networks when linear regression will do!

# Machine Learning in Physics: Real Applications

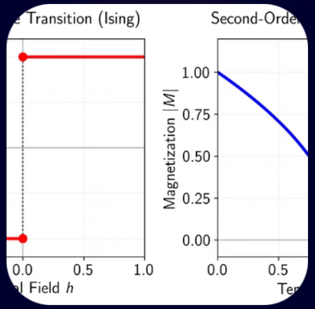
Machine learning is already transforming physics research across multiple domains. Let's explore compelling examples where ML provides capabilities beyond traditional analytical approaches.



## High-Energy Physics

**Particle identification:** Neural networks classify particles (electrons, muons, jets) from detector signatures at CERN's LHC with expert-level accuracy.

**Anomaly detection:** Unsupervised methods discover new physics by flagging unusual collision events that don't fit Standard Model predictions.



## Condensed Matter

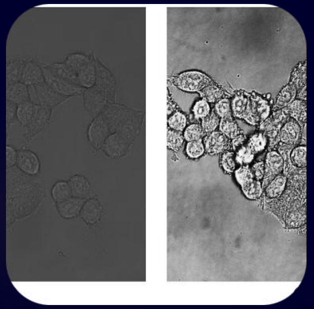
**Phase classification:** CNNs identify phases of matter (ferromagnetic, paramagnetic, topological) from Monte Carlo simulation snapshots, even discovering phase boundaries without being told transition temperatures.

**Quantum states:** Represent many-body wavefunctions with neural networks, enabling new computational approaches to quantum problems.

## Astronomy & Cosmology

**Galaxy classification:** Automated morphology labeling (spiral, elliptical, irregular) from telescope images, processing millions of galaxies for large surveys.

**Gravitational lensing:** Detect subtle lensing effects and infer dark matter distributions from distorted background galaxies.



## Biophysics & Microscopy

**Image segmentation:** Automatically identify and quantify cellular structures, tissue organization, and subcellular components in complex microscopy images.

**Protein structure:** AlphaFold and similar methods predict 3D protein structures from amino acid sequences with remarkable accuracy.

These applications share common themes: **high-dimensional data**, **complex patterns** difficult to encode manually, and **large datasets** from experiments or simulations. Consider how ML might accelerate your own research!

# Common Pitfalls & Best Practices

Even experienced researchers make mistakes when applying machine learning. Being aware of these pitfalls will save you weeks of debugging and prevent publishing flawed results.

### ❌ Data Leakage

**The sin:** Information from validation/test sets influences training.

**Common causes:**

- Scaling/normalizing before splitting (uses test statistics!)
- Feature selection using entire dataset
- Temporal data: training on future, testing on past

**Prevention:** Split first, preprocess second. Fit scalers only on training data.

### ❌ Test Set Abuse

**The sin:** Tuning hyperparameters or making decisions based on test performance.

**Why it matters:** Every time you peek at test results and adjust, you're implicitly training on it. Your reported accuracy becomes optimistically biased.

**Prevention:** Touch test set once, after all development. Use validation for tuning.

### ❌ Overfitting to Noise

**The sin:** Model memorizes random fluctuations, mistakes noise for signal.

**Symptoms:** Training accuracy 99%, validation accuracy 60%.

**Prevention:** Regularization, cross-validation, simpler models, more data. Plot learning curves (train vs. validation error over training).

### ❌ Ignoring Uncertainty

**The sin:** Reporting point predictions without confidence intervals or error bars.

**Why it matters:** Physics requires quantifying uncertainty. ML predictions should include error estimates.

**Solution:** Bootstrap, conformal prediction, Bayesian methods, ensemble variance.

### ✓ Reproducibility Checklist

- Set random seeds everywhere (NumPy, PyTorch, TensorFlow, sklearn)
- Version control your code (Git)
- Document environment (requirements.txt, environment.yml)
- Save data splits and preprocessing steps
- Log hyperparameters and training curves (MLflow, Weights & Biases)
- Convert notebooks to scripts for final runs
- Archive trained models with version tags
- Write clear README with reproduction instructions

**Pro tip:** Create a project template with these practices baked in. Future you (and your collaborators) will thank present you!