Introduction to Machine Learning

For Research Applications

3-Hour Workshop

Comprehensive Guide to ML Fundamentals

Yash Kavaiya

Workshop Agenda

Part 1: Foundations

- ML Overview & Concepts
- Supervised vs Unsupervised
- Key Terminology
- Scikit-learn Introduction

Part 2: Supervised Learning

- Linear Regression
- Logistic Regression
- Decision Trees
- Random Forests

Part 3: Unsupervised Learning

- K-Means Clustering
- Practical Applications

Part 4: Evaluation

- Regression Metrics
- Classification Metrics
- Confusion Matrix
- Cross-Validation

What is Machine Learning?

Definition

Machine Learning is a subset of Artificial Intelligence that enables systems to learn and improve from experience without being explicitly programmed.

Traditional Programming:

- Rules are explicitly coded
- Logic defined by programmer
- Limited adaptability

Machine Learning:

- System learns from data
- Patterns discovered automatically
- Adapts to new data

Data + Algorithm = Model

ML in Research Applications

Scientific Research:

- Drug discovery
- Climate modeling
- Genomics analysis
- Physics simulations

Social Sciences:

- Sentiment analysis
- Behavioral prediction
- Market research
- Policy analysis

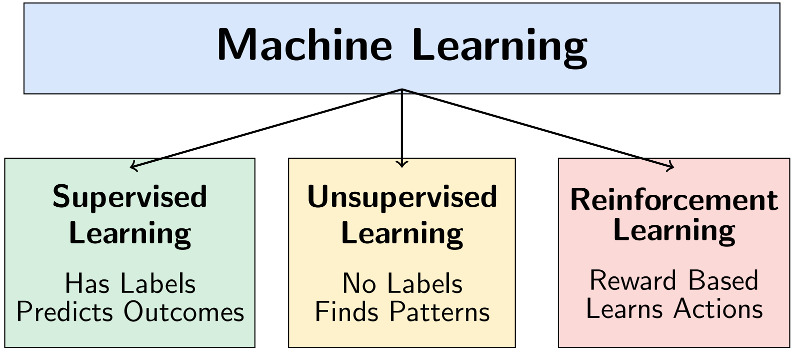
Engineering:

- Predictive maintenance
- Quality control
- Optimization problems
- Fault detection

Healthcare:

- Disease diagnosis
- Treatment planning
- Medical imaging
- Patient outcomes

Types of Machine Learning



Today's focus: Supervised and Unsupervised Learning

Supervised Learning

Definition

Learning from labeled data where the correct answer (target/label) is known during training.

Key Characteristics:

- Training data includes input features AND output labels
- Goal: Learn mapping from inputs to outputs
- Can make predictions on new, unseen data

Regression Tasks:

- Predicting continuous values
- House prices
- Temperature forecasting

Classification Tasks:

- Predicting discrete categories
- Email spam detection
- Disease diagnosis

Supervised Learning: Example

Example: Predicting House Prices

Size (sqft)	Bedrooms	Location	Price ($)
1500	3	Urban	300,000
2000	4	Suburban	400,000
1200	2	Rural	200,000
2500	5	Urban	550,000

- **Features (X):** Size, Bedrooms, Location
- **Label (y):** Price
- **Goal:** Given new house features, predict its price

The model learns the relationship between features and price!

Unsupervised Learning

Definition

Learning from unlabeled data to discover hidden patterns, structures, or relationships.

Key Characteristics:

- No labeled outputs provided
- Goal: Discover underlying structure in data
- Exploratory data analysis

Common Tasks:

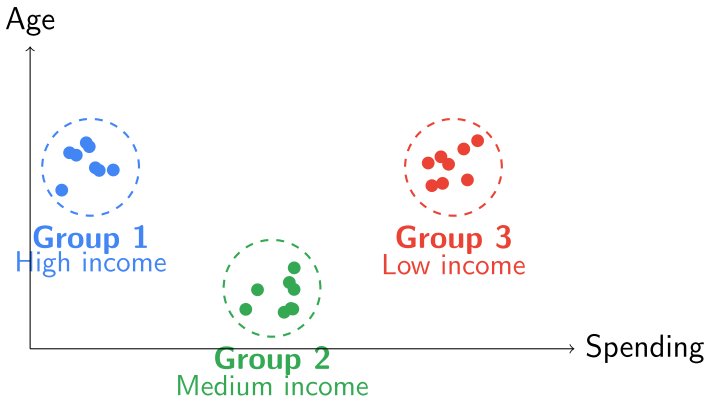
- Clustering (grouping)
- Dimensionality reduction
- Anomaly detection
- Association rules

Applications:

- Customer segmentation
- Topic modeling
- Fraud detection
- Gene sequencing

Unsupervised Learning: Example

Example: Customer Segmentation



Supervised vs Unsupervised: Comparison

Aspect	**Supervised**	**Unsupervised**
Data Type	Labeled (X, y)	Unlabeled (X only)
Goal	Predict outputs	Discover patterns
Training	Learn from examples	Find structure
Evaluation	Compare predictions to actual	Harder to evaluate
Examples	Regression, Classification	Clustering, Reduction
Use Case	When outcomes known	Exploration, grouping

Both are essential tools in the ML researcher's toolkit!

Essential ML Terminology - Part 1

Input variables or attributes used to make predictions. Also called: predictors, independent variables, covariates.

Labels (y)

Output variable we want to predict. Also called: target, dependent variable, response.

Training Data

Dataset used to teach the model. Contains both features and labels (in supervised learning).

Testing Data

Essential ML Terminology - Part 2

Mathematical representation learned from data. Maps inputs to outputs.

Training

Process of learning patterns from training data. The model adjusts its parameters.

Prediction/Inference

Using trained model to make predictions on new, unseen data.

Parameters

Internal variables the model learns during training (e.g., weights in linear regression).

Essential ML Terminology - Part 3

Model learns training data too well, including noise. Performs poorly on new data.

Underfitting

Model is too simple to capture underlying patterns. Performs poorly on both training and test data.

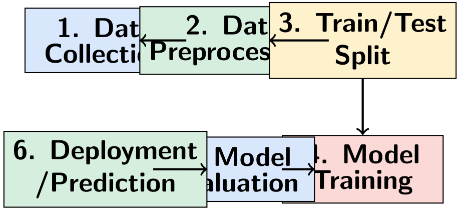
Validation Set

Subset of data used to tune hyperparameters and prevent overfitting during training.

Hyperparameters

Settings configured before training (e.g., learning rate, number of trees). Not learned from data.

The ML Pipeline



Understanding this pipeline is crucial for successful ML projects!

What is Scikit-learn?

Overview

Scikit-learn is Python's most popular machine learning library, providing simple and efficient tools for data mining and analysis.

Key Features:

- Built on NumPy, SciPy, and Matplotlib
- Consistent API across all algorithms
- Extensive documentation and examples
- Free and open-source (BSD license)

What's Included:

- Classification, Regression, Clustering algorithms
- Data preprocessing and feature engineering tools

Installing Scikit-learn

Installation:

```
# Using pip
pip install scikit-learn

# Using conda
conda install scikit-learn

# Verify installation
python -c "import sklearn; print(sklearn.__version__)"
```

Required Dependencies:

- NumPy (¿= 1.17.3)
- SciPy (¿= 1.3.2)

Data Preprocessing: StandardScaler

Why Scale Data?

- Many ML algorithms sensitive to feature scale
- Features with larger ranges dominate the learning
- Standardization improves convergence and performance

StandardScaler: Transforms features to have mean=0 and std=1

```
from sklearn.preprocessing import StandardScaler
import numpy as np

# Sample data
X = np.array([[1, 2], [3, 4], [5, 6]])

# Create and fit scaler
scaler = StandardScaler()
```

Data Preprocessing: LabelEncoder

Why Encode Labels?

- ML algorithms work with numbers, not text
- Convert categorical labels to numeric format
- Essential for classification tasks

LabelEncoder: Converts categories to integers (0, 1, 2, ...)

```
from sklearn.preprocessing import LabelEncoder

# Categorical labels
labels = ['cat', 'dog', 'cat', 'bird', 'dog']

# Create and fit encoder
encoder = LabelEncoder()
encoded_labels = encoder.fit_transform(labels)
```

Train-Test Split

Why Split Data?

- Evaluate model on unseen data (generalization)
- Prevent overfitting
- Simulate real-world performance

Common Split Ratios:

- 80/20 (80% train, 20% test)
- 70/30 or 75/25 for smaller datasets

```
from sklearn.model_selection import train_test_split

# X: features, y: labels
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,       # 20% for testing
```

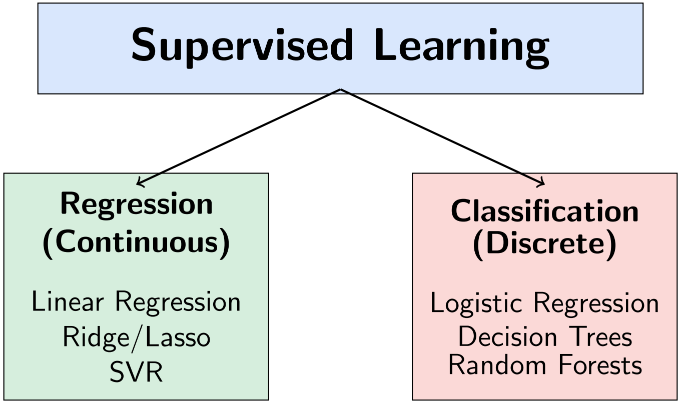
Complete Preprocessing Example

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Load data
df = pd.read_csv('research_data.csv')
X = df.drop('target', axis=1)  # Features
y = df['target']                # Labels

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

Supervised Learning: Overview



Today we'll cover the most common algorithms in each category

Linear Regression: Concept

Definition

Linear Regression models the relationship between input features and a continuous output using a linear equation.

Mathematical Form:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilon$$

Where:

- y = predicted value (dependent variable)
- x_i = input features (independent variables)
- β_i = coefficients (learned parameters)
- β_0 = intercept (bias term)
- ϵ = error term

Linear Regression: Use Cases

When to Use Linear Regression:

- Predicting continuous numerical values
- Relationship between variables appears linear
- Need interpretable results

Research Applications:

- Predicting experiment outcomes
- Dose-response relationships
- Economic forecasting
- Climate predictions

Common Examples:

- House price prediction
- Sales forecasting
- Student performance
- Risk assessment

Assumptions: Linearity, independence, homoscedasticity, normality

Linear Regression: Implementation

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Create model
model = LinearRegression()

# Train model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate
```

Logistic Regression: Concept

Definition

Logistic Regression is used for binary classification, predicting probability of an instance belonging to a particular class.

Sigmoid Function:

$$P(y=1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n)}}$$

Key Points:

- Output: Probability between 0 and 1
- Decision boundary: Typically 0.5 threshold
- Despite the name, it's for classification, not regression!

Binary Classification: Yes/No, True/False, 0/1

Logistic Regression: Use Cases

When to Use Logistic Regression:

- Binary classification problems
- Need probability estimates
- Want interpretable model
- Baseline model for comparison

Research Examples:

- Disease presence/absence
- Success/failure of treatment
- Pass/fail outcomes
- Species identification

Business Examples:

- Customer churn
- Email spam detection
- Fraud detection
- Loan default prediction

Advantage: Provides probability scores, not just class labels

Logistic Regression: Implementation

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,
   classification_report

# Create model
model = LogisticRegression(random_state=42)

# Train model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)
```

Decision Trees: Concept

Definition

Decision Trees make predictions by learning decision rules from features, creating a tree-like model of decisions.

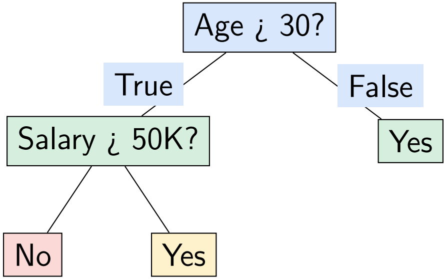
How They Work:

- Split data based on feature values
- Create hierarchical decision rules
- Each internal node = decision on a feature
- Each leaf = prediction/class label

Advantages:

- Easy to understand and visualize
- No feature scaling required
- [illegible]

Decision Trees: Visual Example



Example: Predicting loan approval

- Root: Check age
- If age ¿ 30, check salary
- Leaf nodes: Final decision (Approve/Reject)

Decision Trees: Implementation

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

# Create model
model = DecisionTreeClassifier(
    max_depth=5,          # Limit tree depth
    min_samples_split=10, # Min samples to split
    random_state=42
)

# Train model
model.fit(X_train, y_train)
```

Random Forests: Concept

Definition

Random Forest is an ensemble method that combines multiple decision trees to make more accurate and stable predictions.

How It Works:

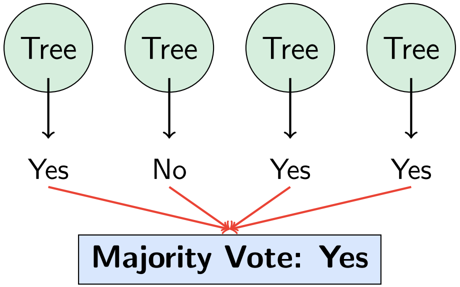
- Creates many decision trees (forest)
- Each tree trained on random subset of data
- Each split considers random subset of features
- Final prediction: Majority vote (classification) or average (regression)

Key Advantages:

- More accurate than single decision tree
- Reduces overfitting

Random Forests: Why They Work

Ensemble Learning Principle



Wisdom of the Crowd: Many weak learners = one strong learner!

Random Forests: Implementation

```python
from sklearn.ensemble import RandomForestClassifier

# Create model
model = RandomForestClassifier(
    n_estimators=100,     # Number of trees
    max_depth=10,         # Max depth of trees
    min_samples_split=5,
    random_state=42,
    n_jobs=-1             # Use all CPU cores
)

# Train model
model.fit(X_train, y_train)
```

K-Means Clustering: Concept

Definition

K-Means is an unsupervised algorithm that groups data points into K clusters based on feature similarity.

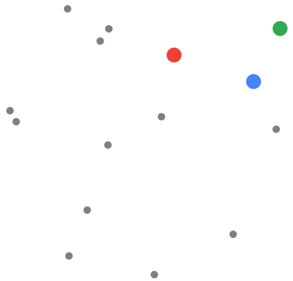
Algorithm Steps:

1. Choose number of clusters (K)
2. Randomly initialize K cluster centers
3. Assign each point to nearest center
4. Update centers to mean of assigned points
5. Repeat steps 3-4 until convergence

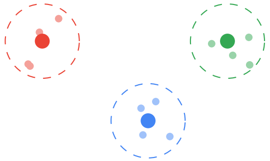
Goal: Minimize within-cluster variance (distance from points to center)

K-Means: Visual Example

Step 1: Initial Random Centers



Step 5: Final Clusters



K-Means: Choosing K

How to Choose Number of Clusters?

1. **Elbow Method:**
   - Plot inertia (sum of squared distances) vs K
   - Look for "elbow" point where improvement slows
   - Balance between fit and complexity
2. **Silhouette Score:**
   - Measures how similar point is to its cluster vs other clusters
   - Range: -1 to 1 (higher is better)
   - Choose K with highest average silhouette score
3. **Domain Knowledge:**
   - Use prior knowledge about data structure
   - Consider practical constraints

K-Means: Implementation

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Create model
kmeans = KMeans(
    n_clusters=3,          # Number of clusters
    random_state=42,
    n_init=10              # Number of initializations
)

# Fit model
kmeans.fit(X)

# Get predictions
```

Finding Optimal K

```
from sklearn.metrics import silhouette_score

# Test different K values
inertias = []
silhouette_scores = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertias.append(kmeans.inertia_)
    silhouette_scores.append(
        silhouette_score(X, kmeans.labels_)
    )
```

K-Means: Applications in Research

Biology & Medicine:

- Gene expression analysis
- Patient stratification
- Disease subtypes

Social Sciences:

- Customer segmentation
- Survey response grouping
- Behavioral patterns

Physical Sciences:

- Image segmentation
- Anomaly detection
- Data compression

Why Model Evaluation Matters

Critical Question

How do we know if our model is actually good?

Key Considerations:

- Training accuracy alone is misleading
- Need to evaluate on unseen data
- Different metrics for different problems
- Balance between different types of errors

Two Main Categories:

- **Regression Metrics:** For continuous predictions
- **Classification Metrics:** For categorical predictions

[illegible]

Regression Metrics: RMSE

Measures average magnitude of prediction errors in same units as target variable.

Formula:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Where: y_i = actual value, $\hat{y}_i$ = predicted value

Characteristics:

- Penalizes large errors more heavily (due to squaring)
- Same units as target variable (interpretable)
- Lower is better (0 = perfect predictions)

Regression Metrics: MAE

Measures average absolute difference between predictions and actual values.

Formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Characteristics:

- Treats all errors equally (no squaring)
- Same units as target variable
- Lower is better $(0 = \text{perfect})$
- More robust to outliers than RMSE

Regression Metrics: Implementation

```python
from sklearn.metrics import mean_squared_error,
    mean_absolute_error, r2_score
import numpy as np

# Make predictions
y_pred = model.predict(X_test)

# Calculate metrics
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"RMSE: {rmse:.2f}")
print(f"MAE: {mae:.2f}")
```

Classification Metrics: Accuracy

Proportion of correct predictions among total predictions.

Formula:

$$Accuracy = \frac{Correct\ \ Predictions}{Total\ \ Predictions} = \frac{TP + TN}{TP + TN + FP + FN}$$

When to Use:

- Balanced classes (similar number in each category)
- All errors are equally important

Warning: Can be misleading with imbalanced classes!

Example: In fraud detection with 99% non-fraud, a model that always predicts "not fraud" has 99% accuracy but is useless!

Classification Metrics: Precision & Recall

Of all positive predictions, how many were correct?

Recall

Of all actual positives, how many did we find?

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Use when:

- False positives are costly
- Example: Spam detection

Use when:

- False negatives are costly
- Example: Disease detection

Trade-off: High precision often means lower recall, and vice versa

Classification Metrics: F1-Score

Harmonic mean of Precision and Recall, balancing both metrics.

Formula:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Characteristics:

- Range: 0 to 1 (higher is better)
- Best when you need balance between precision and recall
- Useful for imbalanced datasets
- Penalizes extreme values (very high precision but low recall)

When to Use: When you care about both false positives AND false negatives

Confusion Matrix

Visual Representation of Classification Performance

		Predicted	
		Positive	**Negative**
Actual	**Positive**	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

- **TP:** Correctly predicted positive
- **TN:** Correctly predicted negative
- **FP:** Incorrectly predicted positive (Type I error)
- **FN:** Incorrectly predicted negative (Type II error)

Classification Metrics: Implementation

```python
from sklearn.metrics import (accuracy_score, precision_score,
                             recall_score, f1_score,
                             confusion_matrix,
                                classification_report)

# Make predictions
y_pred = model.predict(X_test)

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

Visualization: Confusion Matrix

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Create confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Negative', 'Positive'],
            yticklabels=['Negative', 'Positive'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
```

Why Cross-Validation?

Problem with Single Train-Test Split

Results can vary significantly depending on how you split the data. A lucky/unlucky split gives misleading performance estimates.

Solution: Cross-Validation

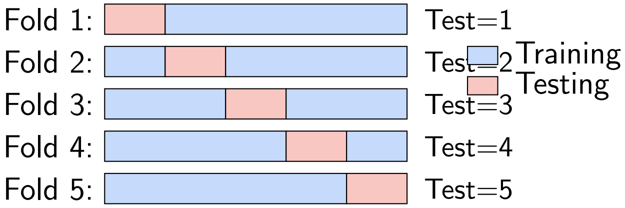
- Use multiple train-test splits
- Average results across all splits
- More robust estimate of model performance
- Better use of limited data

Benefits:

- Reduces variance in performance estimates
- Detects overfitting more reliably

K-Fold Cross-Validation

Split data into K equal parts (folds). Train on K-1 folds, test on remaining fold. Repeat K times, each time with different test fold.



Common Choice: K=5 or K=10

Cross-Validation: Implementation

```python
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

# Create model
model = LogisticRegression()

# Perform 5-fold cross-validation
scores = cross_val_score(model, X, y, cv=5,
                         scoring='accuracy')

print("Cross-Validation Scores:", scores)
print(f"Mean Accuracy: {scores.mean():.3f}")
print(f"Std Dev: {scores.std():.3f}")
```

Stratified K-Fold

With imbalanced classes, random splits might not preserve class distribution in each fold.

Stratified K-Fold Solution:

- Maintains class distribution in each fold
- Each fold has approximately same percentage of each class
- Particularly important for imbalanced datasets

Example:

- Original data: 80% Class A, 20% Class B
- Each fold will have approximately 80% Class A, 20% Class B
- More reliable performance estimates

Default in scikit-learn: Classification tasks automatically use stratified splits

Stratified Cross-Validation

```python
from sklearn.model_selection import StratifiedKFold

# Create stratified k-fold
skf = StratifiedKFold(n_splits=5, shuffle=True,
                      random_state=42)

# Manual cross-validation with stratification
scores = []
for train_idx, test_idx in skf.split(X, y):
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
```

Cross-Validation: Best Practices

Choosing K:

- K=5 or K=10 most common
- Larger K = more training data per fold (less bias)
- Larger K = more computational cost
- Leave-One-Out (K=n): Maximum training data, very expensive

Important Considerations:

- Always use stratification for classification
- Shuffle data before splitting (with fixed random_state)
- Don't use for hyperparameter tuning (use nested CV)
- Still need separate test set for final evaluation

Remember: CV for model selection, held-out test set for final performance report

Complete ML Pipeline Example - Part 1

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# 1. Load data
df = pd.read_csv('research_data.csv')
X = df.drop('target', axis=1)
y = df['target']

# 2. Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
```

Complete ML Pipeline Example - Part 2

```
# 4. Train model
model = RandomForestClassifier(n_estimators=100,
                               random_state=42)
model.fit(X_train_scaled, y_train)

# 5. Evaluate
y_pred = model.predict(X_test_scaled)
print(classification_report(y_test, y_pred))

# 6. Cross-validation
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(model, X_train_scaled,
                            y_train, cv=5)
print(f"\nCV Mean Accuracy: {cv_scores.mean():.3f}")
```

Workshop Summary

What We Covered Today:

Foundations:

- ML concepts & terminology
- Supervised vs Unsupervised
- Scikit-learn basics
- Data preprocessing

Supervised Learning:

- Linear Regression
- Logistic Regression
- Decision Trees
- Random Forests

Unsupervised Learning:

- K-Means clustering
- Choosing optimal K
- Practical applications

Evaluation:

- Regression metrics
- Classification metrics
- Confusion matrix
- Cross-validation

Key Takeaways

Machine Learning is a powerful tool for research, but requires careful application and evaluation.

Best Practices:

1. Always split data before any preprocessing
2. Use appropriate metrics for your problem
3. Validate with cross-validation
4. Check for overfitting/underfitting
5. Interpret results in context of your research question

Next Steps:

- Practice with your own research data

Learning Resources

Official Documentation:

- Scikit-learn: https://scikit-learn.org
- Pandas: https://pandas.pydata.org
- NumPy: https://numpy.org

Books:

- "Hands-On Machine Learning" - Aurélien Géron
- "Introduction to Statistical Learning" - James et al.
- "Python Machine Learning" - Sebastian Raschka

Online Courses:

- Coursera: Machine Learning by Andrew Ng
- Fast.ai: Practical Deep Learning
- DataCamp: Machine Learning Track

Practice Datasets

Start with Standard Datasets:

Built into Scikit-learn:

- Iris dataset (classification)
- Boston housing (regression)
- Wine quality (classification)

Online Repositories:

- UCI Machine Learning Repository
- Kaggle Datasets
- Google Dataset Search
- OpenML

Your Own Research:

- Apply to your research problems

Thank You!

Questions & Discussion

Stay Connected

Yash Kavaiya

LinkedIn:	linkedin.com/in/yashkavaiya
YouTube:	youtube.com/@genai-guru
Website:	easy-ai-labs.lovable.app
Company:	Gen AI Guru