Lab 1 — Your First ML Experiment (Iris Classification)

Time: 60–90 minutes
Difficulty: Beginner
What you'll learn

- Frame an ML problem (features, target, metric)
- Establish a baseline (why it matters)
- Train a simple model (k-Nearest Neighbors) in a reproducible pipeline
- Evaluate with accuracy, precision/recall/F1, confusion matrix
- Run a quick hyperparameter search and compare results

Prerequisites

- A Google account (for Colab)
- Basic Python familiarity (variables, lists) — we'll still show every step
- Stable internet

> If you prefer local: install Python 3.9+, then `pip install scikit-learn matplotlib pandas numpy`.

Deliverables

1. A Colab notebook (.ipynb) with all cells executed.

2. A short reflection (5–8 bullet points) answering the questions at the end.

Step-by-step Instructions

Step 0 — Open and prepare your notebook (Colab)

1. Go to colab.research.google.com → **New Notebook**.
2. Rename it to: `ML-AI-Foundations-Section1-Lab1-YourName`.
3. In **Runtime → Change runtime type**, ensure **Python 3**.

Step 1 — Set up imports and a reproducible environment

Add a new code cell and run:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.dummy import DummyClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_m
import matplotlib.pyplot as plt
import random

# Reproducibility
SEED = 42
np.random.seed(SEED)
random.seed(SEED)
```

What this does: imports the tools you need and fixes seeds so results are consistent.

Step 2 — Load and inspect the dataset

Add a code cell:

```
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target, name="species")

print("Features shape:", X.shape)
print("Target shape:", y.shape)
X.head()
```

Expect: 150 rows × 4 columns. Target has 3 classes (0,1,2) → iris setosa, versicolor, virginica.

Optional: map numbers to names.

```
target_names = dict(enumerate(iris.target_names))
y_named = y.map(target_names)
y_named.head()
```

Step 3 — Quick visualization (intuition building)

Add a code cell:

```
plt.figure(figsize=(6,4))
plt.scatter(X['sepal length (cm)'], X['petal length (cm)'], c=y, alpha=0.8)
plt.xlabel('Sepal length (cm)')
plt.ylabel('Petal length (cm)')
plt.title('Iris: Sepal length vs Petal length (colored by class)')
plt.show()
```

Goal: see that classes are somewhat separable (especially setosa).

Step 4 — Frame the ML problem

- **Type:** Multi-class classification
- **Features (X):** 4 numeric measurements
- **Target (y):** species (3 classes)
- **Primary metric: Accuracy** (simple starting point)
- **Baselines:** "Most frequent class" accuracy to beat

Add a markdown cell in your notebook summarizing the above in your own words.

Step 5 — Train/test split (with stratification)

Add a code cell:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=SEED, stratify=y
)

print("Train size:", X_train.shape[0], " Test size:", X_test.shape[0])
```

Why stratify? Keeps class balance similar in train and test.

Step 6 — Establish a baseline

Add a code cell:

```
baseline = DummyClassifier(strategy="most_frequent", random_state=SEED)
baseline.fit(X_train, y_train)
y_pred_base = baseline.predict(X_test)
```

```
base_acc = accuracy_score(y_test, y_pred_base)
print(f"Baseline accuracy (most frequent class): {base_acc:.3f}")
```

Takeaway: Any real model should beat this.

Step 7 — Build a proper ML pipeline and train KNN

We'll **scale** features (important for distance-based models) and then fit **KNN**.

Add a code cell:

```
knn_pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("knn", KNeighborsClassifier(n_neighbors=5))
])

knn_pipeline.fit(X_train, y_train)
y_pred = knn_pipeline.predict(X_test)

acc = accuracy_score(y_test, y_pred)
print(f"KNN (k=5) accuracy: {acc:.3f}")

print("\nClassification report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))
```

Interpretation tips (add to a markdown cell):

- **Precision:** Of predicted class A, how many were actually A?
- **Recall:** Of actual class A, how many did we find?
- **F1:** Balance between precision & recall (per class).

Step 8 — Try a small hyperparameter search (GridSearchCV)

Let's see how different **k** values affect performance (using 5-fold cross-validation).

Add a code cell:

```
param_grid = {"knn__n_neighbors": list(range(1, 16))}

grid = GridSearchCV(
    estimator=knn_pipeline,
    param_grid=param_grid,
    cv=5,
    n_jobs=-1
)

grid.fit(X_train, y_train)
print("Best CV params:", grid.best_params_)
print("Best CV score:", grid.best_score_)

# Evaluate best model on the test set
best_model = grid.best_estimator_
y_pred_best = best_model.predict(X_test)
acc_best = accuracy_score(y_test, y_pred_best)
print(f"Test accuracy with best k: {acc_best:.3f}")
print("\nConfusion matrix:\n", confusion_matrix(y_test, y_pred_best))
```

Discussion prompt: Did cross-validation pick a different **k**? Did test accuracy change meaningfully?

Step 9 — Minimal error analysis

Which samples were misclassified?

Add a code cell:

```
mis_idx = np.where(y_test != y_pred_best)[0]
print("Misclassified indices (in test set order):", mis_idx)
```

```python
if len(mis_idx) > 0:
    # Show their feature rows and true/pred labels
    err_rows = X_test.iloc[mis_idx].copy()
    err_rows['true'] = y_test.iloc[mis_idx].values
    err_rows['pred'] = y_pred_best[mis_idx]
    err_rows['true_name'] = err_rows['true'].map(target_names)
    err_rows['pred_name'] = err_rows['pred'].map(target_names)
    display(err_rows)
else:
    print("No misclassifications this run.")
```

Goal: See where the model struggles (e.g., versicolor vs virginica).

Step 10 — (Optional stretch) Compare with Logistic Regression

Add a code cell:

```python
from sklearn.linear_model import LogisticRegression

logreg_pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("lr", LogisticRegression(max_iter=500, multi_class="auto"))
])

logreg_pipeline.fit(X_train, y_train)
y_pred_lr = logreg_pipeline.predict(X_test)
print("LogReg test accuracy:", accuracy_score(y_test, y_pred_lr))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred_lr))
```

Prompt: Which model did better on this split? Why might that be?

Step 11 — Save your work

- **File → Save a copy in Drive**

- **File → Download → Download .ipynb** and upload to your course portal if required.