**Week-2 Summary**

**Theme:** Read a CSV dataset, sanity-check it with basic stats, and evaluate a simple set of if-then rules for practicing both **pandas** and **NumPy**.

Notebook to run: <https://colab.research.google.com/drive/1WM8U9bvDQjRjXeoozruFqp-xe3_tQqXF?usp=sharing>

**1) Get the data (Iris, UCI Repository)**

* Download the Iris dataset ZIP from the UCI link and extract iris.data, then drag-and-drop it into your Colab runtime.
* (Alternative) You can also read directly from a URL in pandas — shown below.

**2) Load the CSV in pandas**

Download the dataset zip file and extract it. Start a new notebook in Google Colab and then you can copy (drag&drop) the iris.data file to Colab’s home directory for your runtime.

Link: https://archive.ics.uci.edu/dataset/53/iris

import pandas as pd

column\_names = ["sepal\_length",

"sepal\_width",

"petal\_length",

"petal\_width",

"class"]

# Option A: local file you uploaded to Colab

X = pd.read\_csv("iris.data", header=None, names=column\_names)

# Option B: directly from URL (works too)

X = pd.read\_csv(

"https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data",

header=None,

names=column\_names

)

X.head(10) # preview first 10 rows

**Note:** Iris has **no header row**, so we pass header=None and set names ourselves (find the correct order from the file named iris.names in the zip file).

**3) Separate features and labels**

y = X["class"] # Series

X = X.drop(columns=["class"]) # DataFrame with 4 numeric columns

X.shape, y.shape # -> ((150, 4), (150,))

Note that X is a DataFrame (2-dimensional, just like a csv file typically is, something tabular) and y is a Series (1-dimensional, it’s just like a bunch of strings in a list).

**4) Quick sanity checks**

# For numeric features

X.describe() # min, max, quartiles, mean, std, count

# For labels

y = y.astype("category")

y.cat.categories # category names

y.value\_counts() # class counts (support per class)

# y.describe() shows count, unique, top, freq for a categorical series

**NaN handling:** If a column has missing values, X.describe() will show reduced count. Use X.isna().sum() to check.

**Types:** Keep features numeric (float), labels categorical/string → then convert to category/codes when needed.

**Force a specific label→code mapping:** If you want make sure that  
 setosa→0, versicolor→1, virginica→2, use

order = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]

y = y.cat.set\_categories(order, ordered=True)

**5) Convert to NumPy (when you want array ops)**

import numpy as np

X\_np = X.values # shape (150, 4)

y\_np = y.cat.codes.values # shape (150,), values in {0,1,2}

#instead of .values you can also use to\_numpy()

#X\_np = X.to\_numpy()

#y\_np = y.cat.codes.to\_numpy()

**6) Rule-based classifier**

**Rules (for practice only):**

1. If **petal\_length < 3.0** → predict **setosa (0)**
2. Else if **sepal\_width < 3.05** → predict **versicolor (1)**
3. Else → predict **virginica (2)**

Note that **petal\_length and sepal\_width** in rules 1 and 2 have column indices of 2 and 1, respectively.

# Column indices for readability

SL, SW, PL, PW = 0, 1, 2, 3

**A) Simple loop version**

y\_pred = []

for row in X\_np:

if row[PL] < 3.0:

cls = 0

elif row[SW] < 3.05:

cls = 1

else:

cls = 2

y\_pred.append(cls)

y\_pred = np.array(y\_pred)

It is a good idea to start with an empty list and then keep appending to it. On the last line above, we convert y\_pred to numpy even though we didn’t have to do that: accuracy\_score accepts lists as well as numpy arrays.

**B) Vectorized (NumPy) version with np.select**

cond1 = X\_np[:, PL] < 3.0

cond2 = X\_np[:, SW] < 3.05 #and cond1 is not satisfied

y\_pred\_vec = np.select(

condlist=[cond1, cond2],

choicelist=[0, 1],

default=2

)

Both y\_pred and y\_pred\_vec should match:

assert np.array\_equal(y\_pred, y\_pred\_vec)

**7) Evaluate accuracy (two ways)**

from sklearn.metrics import accuracy\_score

acc1 = accuracy\_score(y\_np, y\_pred)

is\_correct = (y\_np == y\_pred)

acc2 = is\_correct.mean()

acc1, acc2

**Confusion matrix & Classification report**

This connects back to Week-1 (precision/recall per class).

from sklearn.metrics import confusion\_matrix, classification\_report

cm = confusion\_matrix(y\_np, y\_pred) # rows: true, cols: pred

print(cm)

print(classification\_report(

y\_np, y\_pred,

target\_names=["setosa", "versicolor", "virginica"],

digits=3

))