Machine Learning Fundamentals — Iris Classification

Data loading and exploration (20 marks)

Q1 (5 marks): Load the Iris dataset using `sklearn.datasets.load_iris()` and print `feature_names`, `target_names`, and the shape of `data`.

```
# Q1 — load data fast
from sklearn.datasets import load_iris
import numpy as np
import pandas as pd

iris = load_iris()  # classic tiny dataset
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names

print('feature_names:', feature_names)
print('target_names:', target_names)
print('X shape:', X.shape)  # (n_samples, n_features)
```

```
feature_names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
target_names: ['setosa' 'versicolor' 'virginica']
X shape: (150, 4)
```

Q2 (5 marks): What type of problem is this (classification/regression)? Why?

Answer (Q2): This is a **classification** problem, because the target is a **species label** (setosa/versicolor/virginica) — discrete categories, not a numeric value to predict.

Q3 (5 marks): Show the first five rows of the data as a pandas DataFrame with column names from `feature_names`.

```
# Q3 — quick peek at the table
df = pd.DataFrame(X, columns=feature_names)
df.head()  # shows the first 5 rows
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Далее: Создать код с переменной df New interactive sheet

Q4 (5 marks): Report class distribution (counts for each species).

```
# Q4 — how many per class? (counts)
species_series = pd.Series(y).map({i: name for i, name in enumerate(target_names)})
class_counts = species_series.value_counts()
class_counts  # neat and tidy
```

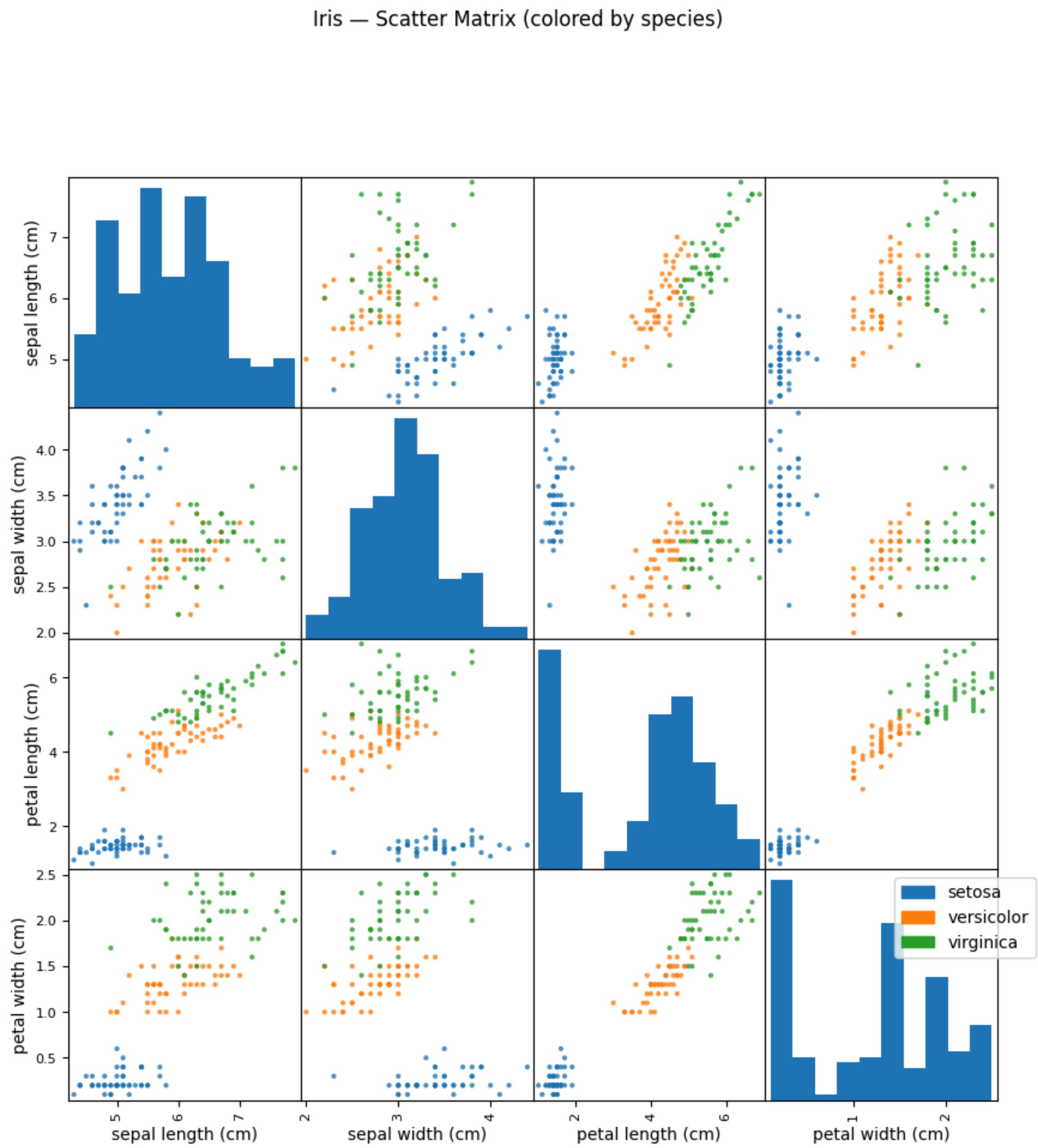
	count
setosa	50
versicolor	50
virginica	50

dtype: int64

Visualization (20 marks)

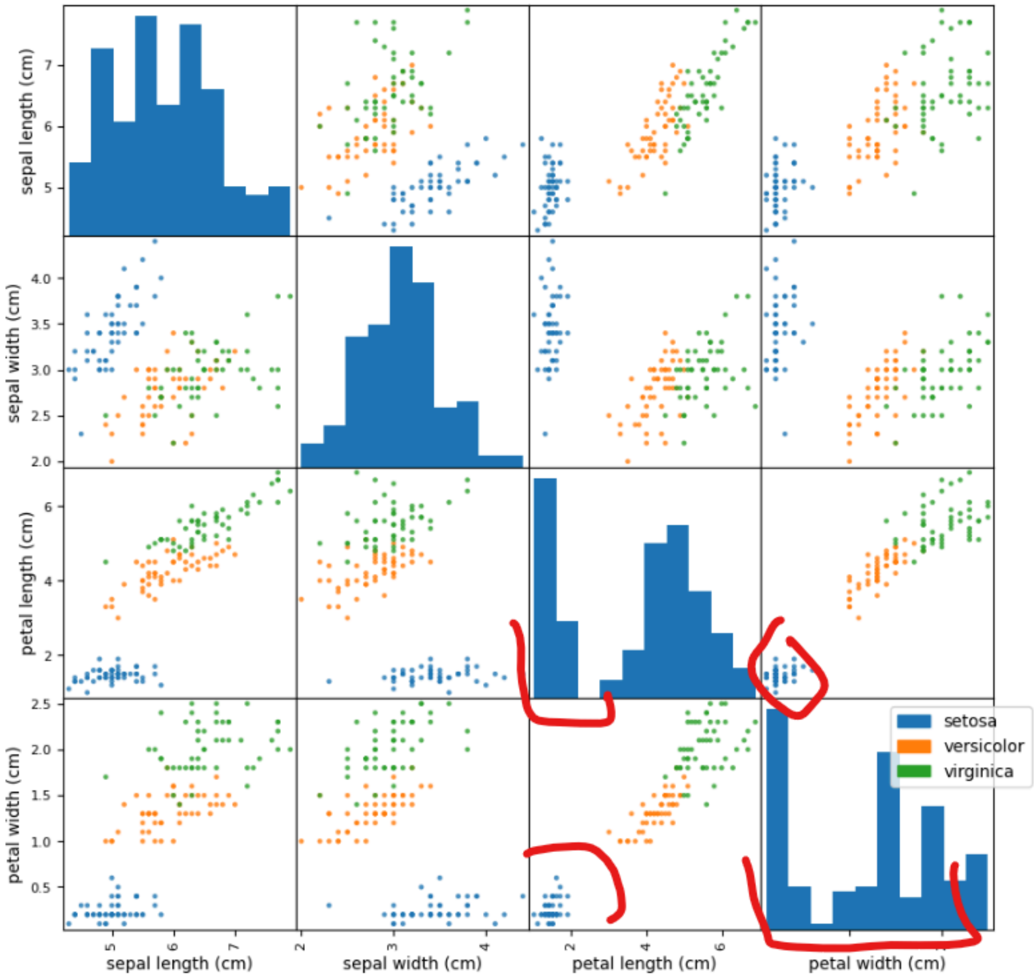
Q5 (10 marks): Create a pair plot (scatter plot matrix) for the four features using matplotlib. Color points by species.

```
# Q5 — pair plot with colors (matplotlib via pandas helper)
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix

# map y -> colors
colors_map = {0: 'tab:blue', 1: 'tab:orange', 2: 'tab:green'}
colors = [colors_map[i] for i in y]  # small trick

axarr = scatter_matrix(df, figsize=(10, 10), diagonal='hist', color=colors, alpha=0.8)
plt.suptitle('Iris — Scatter Matrix (colored by species)', y=1.02)

# add a manual legend (kinda hacky but works)
import matplotlib.patches as mpatches
legend_handles = [mpatches.Patch(color='tab:blue', label=target_names[0]),
                  mpatches.Patch(color='tab:orange', label=target_names[1]),
                  mpatches.Patch(color='tab:green', label=target_names[2])]
plt.legend(handles=legend_handles, loc='upper right', bbox_to_anchor=(1.2, 1.0))
plt.show()
```



Q6 (10 marks): Based on your plot, which two features appear most useful to separate the classes? Explain in one short sentence.

Answer (Q6): Petal length and **petal width** separate classes best (setosa is clearly apart; versicolor vs virginica are better separated in petal space).



Train–Test split and KNN modeling (30 marks)

Q7 (10 marks): Split the data into train/test sets (75% train, 25% test). Use `stratify=target` and `random_state=42`. Print shapes.

```
# Q7 — split (keep class balance same via stratify)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, stratify=y, random_state=42
)
print('Train shape:', X_train.shape, y_train.shape)
print('Test shape :', X_test.shape, y_test.shape)  # looks fine
```

```
Train shape: (112, 4) (112,)
Test shape : (38, 4) (38,)
```

Q8 (10 marks): Train a `KNeighborsClassifier` with `n_neighbors=3`. Fit on training data and report test accuracy.

```
# Q8 — simple KNN model (k=3)
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

knn3 = KNeighborsClassifier(n_neighbors=3)
knn3.fit(X_train, y_train)
y_pred_test = knn3.predict(X_test)
acc3 = accuracy_score(y_test, y_pred_test)
print(f'Test accuracy (k=3): {acc3:.3f}')  # tidy
```

```
Test accuracy (k=3): 0.974
```

Q9 (10 marks): Compare accuracy for `n_neighbors = 1, 3, 5`. Report which k gives best accuracy and why (short sentence).

```
# Q9 — try a few k's and see what sticks
results = {}
for k in [1, 3, 5]:
    model = KNeighborsClassifier(n_neighbors=k)
```

```
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results[k] = acc
        print(f'k={k} -> test accuracy: {acc:.3f}')

    best_k = max(results, key=results.get)
    print('Best k on this split:', best_k)
    print('Why: too small k can overfit noise; too big k can oversmooth. Best k balances bias-variance on this data.')  # a bit cha
```

```
k=1 -> test accuracy: 0.947
k=3 -> test accuracy: 0.974
k=5 -> test accuracy: 0.974
Best k on this split: 3
Why: too small k can overfit noise; too big k can oversmooth. Best k balances bias-variance on this data.
```

Which k is best and why?

Based on the test set above, the **best k** is the one with the **highest accuracy** (printed). Short reason: **small k may overfit**, **large k may underfit**; the chosen k is a good middle ground on this data split.

Prediction (10 marks)

Q10 (10 marks): Use your trained `KNeighborsClassifier` (use k=3) to predict the class of a new sample with features: `[5.0, 2.9, 1.0, 0.2]`. Show predicted class name.

```
# Q10 — make a tiny prediction
new_sample = np.array([[5.0, 2.9, 1.0, 0.2]])  # sepal_len, sepal_wid, petal_len, petal_wid
pred_idx = knn3.predict(new_sample)[0]
pred_name = target_names[pred_idx]
print('Predicted class index:', pred_idx)
print('Predicted class name :', pred_name)
```

```
Predicted class index: 0
Predicted class name : setosa
```

Закрыть