```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
```

```
pd.read_csv('/content/drive/MyDrive/Datasets/IRIS.csv')
ad()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Next steps: Generate code with df | New interactive sheet

```
num_cols = df.select_dtypes(include=[np.number]).columns

df[num_cols] = df[num_cols].fillna(df[num_cols].mean())

print("===== AFTER FILLING MISSING VALUES =====")
display(df.head())

scaler = MinMaxScaler()
df[num_cols] = scaler.fit_transform(df[num_cols])

print("===== AFTER MIN-MAX SCALING =====")
display(df.head())
```

```
===== AFTER FILLING MISSING VALUES =====
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
===== AFTER MIN-MAX SCALING =====
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	0.222222	0.625000	0.067797	0.041667	Iris-setosa
1	0.166667	0.416667	0.067797	0.041667	Iris-setosa
2	0.111111	0.500000	0.050847	0.041667	Iris-setosa
3	0.083333	0.458333	0.084746	0.041667	Iris-setosa
4	0.194444	0.666667	0.067797	0.041667	Iris-setosa

```
split_ratios = [(0.5, 0.5), (0.6, 0.4), (0.7, 0.3), (0.8, 0.2)]
k_values = [1, 3, 5, 7, 9, 11]

X = df.iloc[:, :-1]   # features
y = df.iloc[:, -1]    # target
results = []
```

```
from sklearn.model_selection import train_test_split

# X = features, y = target
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

split_ratios = [(0.5, 0.5), (0.6, 0.4), (0.7, 0.3), (0.8, 0.2)]

print("===== TRAIN / TEST SPLIT OUTPUTS =====\n")

for train_size, test_size in split_ratios:
    X_train, X_test, y_train, y_test = train_test_split(
        X, y,
        train_size=train_size,
```

```
            test_size=test_size,
            random_state=42,
            shuffle=True
        )

        print(f"Split Ratio: {int(train_size*100)} / {int(test_size*100)}")
        print(f"X_train shape: {X_train.shape}")
        print(f"X_test shape:  {X_test.shape}")
        print(f"y_train shape: {y_train.shape}")
        print(f"y_test shape:  {y_test.shape}")
        print("--------------------------------------")
```

```
===== TRAIN / TEST SPLIT OUTPUTS =====

Split Ratio: 50 / 50
X_train shape: (75, 4)
X_test shape:  (75, 4)
y_train shape: (75,)
y_test shape:  (75,)
--------------------------------------
Split Ratio: 60 / 40
X_train shape: (90, 4)
X_test shape:  (60, 4)
y_train shape: (90,)
y_test shape:  (60,)
--------------------------------------
Split Ratio: 70 / 30
X_train shape: (105, 4)
X_test shape:  (45, 4)
y_train shape: (105,)
y_test shape:  (45,)
--------------------------------------
Split Ratio: 80 / 20
X_train shape: (120, 4)
X_test shape:  (30, 4)
y_train shape: (120,)
y_test shape:  (30,)
--------------------------------------
```

```
for train_size, test_size in split_ratios:
    for k in k_values:

        X_train, X_test, y_train, y_test = train_test_split(
            X, y, train_size=train_size, test_size=test_size, random_state=42,
        )

        model = KNeighborsClassifier(n_neighbors=k)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)


        acc = accuracy_score(y_test, y_pred)
        prec = precision_score(y_test, y_pred, average="macro")
```

```
            rec = recall_score(y_test, y_pred, average="macro")
            f1 = f1_score(y_test, y_pred, average="macro")
            cm = confusion_matrix(y_test, y_pred)

            results.append({
                "train_size": train_size,
                "test_size": test_size,
                "k": k,
                "accuracy": acc,
                "precision": prec,
                "recall": rec,
                "f1_score": f1,
                "confusion_matrix": cm
            })
```

```
results_df = pd.DataFrame([
    {
        "train_size": r["train_size"],
        "test_size": r["test_size"],
        "k": r["k"],
        "accuracy": r["accuracy"],
        "precision": r["precision"],
        "recall": r["recall"],
        "f1_score": r["f1_score"]
    }
    for r in results
])

best = results_df.iloc[results_df["f1_score"].idxmax()]

print("===== SUMMARY OF ALL RESULTS =====")
display(results_df)

print("\n===== BEST CONFIGURATION =====")
print(best)
```

Next steps: Generate code with `results_df` New interactive sheet

```
===== SUMMARY OF ALL RESULTS =====
```

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


for r in results:
    plt.figure(figsize=(5,4))
    sns.heatmap(r["confusion_matrix"], annot=True, fmt="d", c
    plt.title(f"Confusion Matrix\nTrain/Test {int(r['train_si
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.tight_layout()
    plt.show()
```

7	0.6	0.4	3	0.983333	0.983333	0.981481	0.981929
8	0.6	0.4	5	0.983333	0.983333	0.981481	0.981929
9	0.6	0.4	7	0.983333	0.983333	0.981481	0.981929
10	0.6	0.4	9	0.983333	0.983333	0.981481	0.981929
11	0.6	0.4	11	0.983333	0.983333	0.981481	0.981929
12	0.7	0.3	1	1.000000	1.000000	1.000000	1.000000
13	0.7	0.3	3	1.000000	1.000000	1.000000	1.000000
14	0.7	0.3	5	1.000000	1.000000	1.000000	1.000000
15	0.7	0.3	7	1.000000	1.000000	1.000000	1.000000
16	0.7	0.3	9	1.000000	1.000000	1.000000	1.000000
17	0.7	0.3	11	1.000000	1.000000	1.000000	1.000000
18	0.8	0.2	1	1.000000	1.000000	1.000000	1.000000
19	0.8	0.2	3	1.000000	1.000000	1.000000	1.000000
20	0.8	0.2	5	1.000000	1.000000	1.000000	1.000000
21	0.8	0.2	7	1.000000	1.000000	1.000000	1.000000
22	0.8	0.2	9	1.000000	1.000000	1.000000	1.000000
23	0.8	0.2	11	1.000000	1.000000	1.000000	1.000000

```
===== BEST CONFIGURATION =====
train_size    0.7
test_size     0.3
k             1.0
accuracy      1.0
precision     1.0
recall        1.0
f1_score      1.0
Name: 12, dtype: float64
```