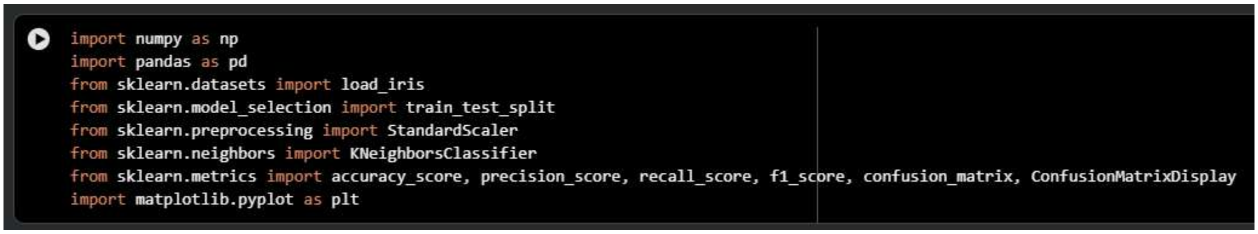
LAB # 08

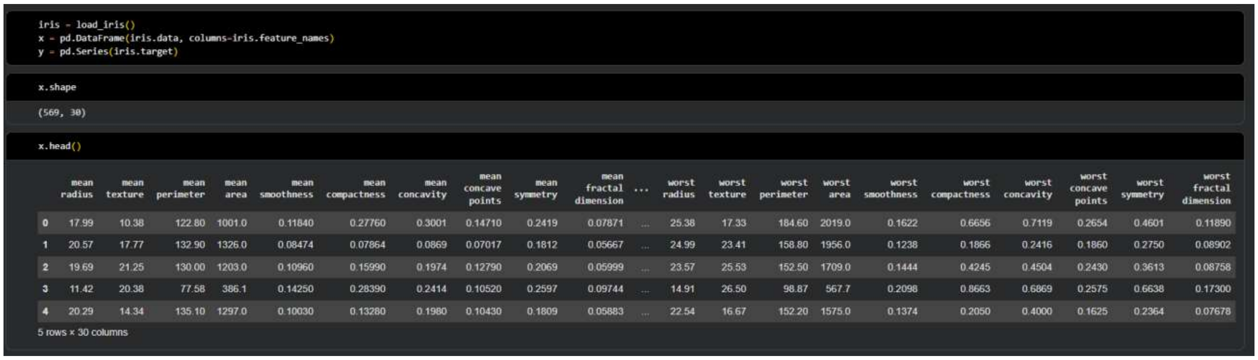
KNN CLASSIFIER

LAB TASKS

1. Load a dataset (e.g., Iris, Breast Cancer dataset).



```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```



```
iris = load_iris()
x = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)
```

```
x.shape
```

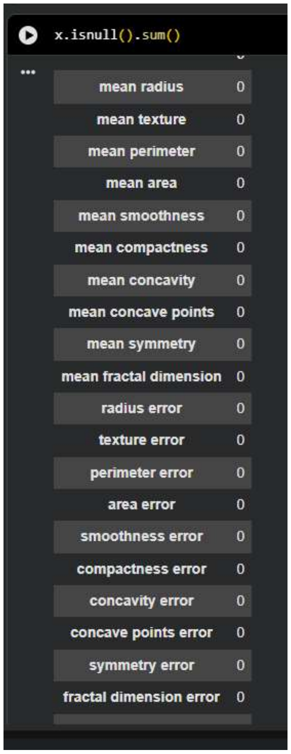
```
(569, 30)
```

```
x.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

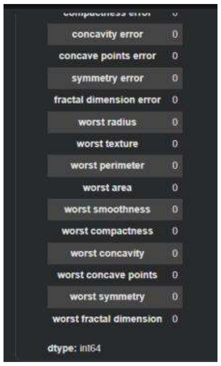
5 rows × 30 columns

2. Apply data preprocessing (handle missing values, encode categorical data).



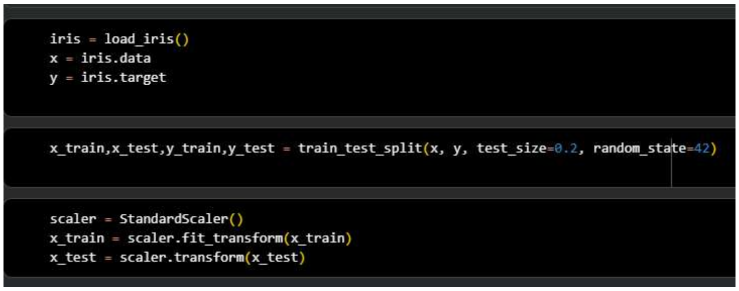
```
x.isnull().sum()
```

mean radius	0
mean texture	0
mean perimeter	0
mean area	0
mean smoothness	0
mean compactness	0
mean concavity	0
mean concave points	0
mean symmetry	0
mean fractal dimension	0
radius error	0
texture error	0
perimeter error	0
area error	0
smoothness error	0
compactness error	0
concavity error	0
concave points error	0
symmetry error	0
fractal dimension error	0



```
compactness error        0
concavity error          0
concave points error     0
symmetry error           0
fractal dimension error  0
worst radius             0
worst texture            0
worst perimeter          0
worst area               0
worst smoothness         0
worst compactness        0
worst concavity          0
worst concave points     0
worst symmetry           0
worst fractal dimension  0

dtype: int64
```

3. Split the dataset into training and testing sets.

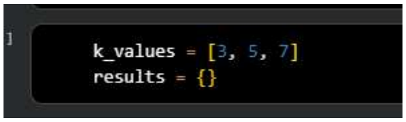


```
iris = load_iris()
x = iris.data
y = iris.target
```

```
x_train,x_test,y_train,y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

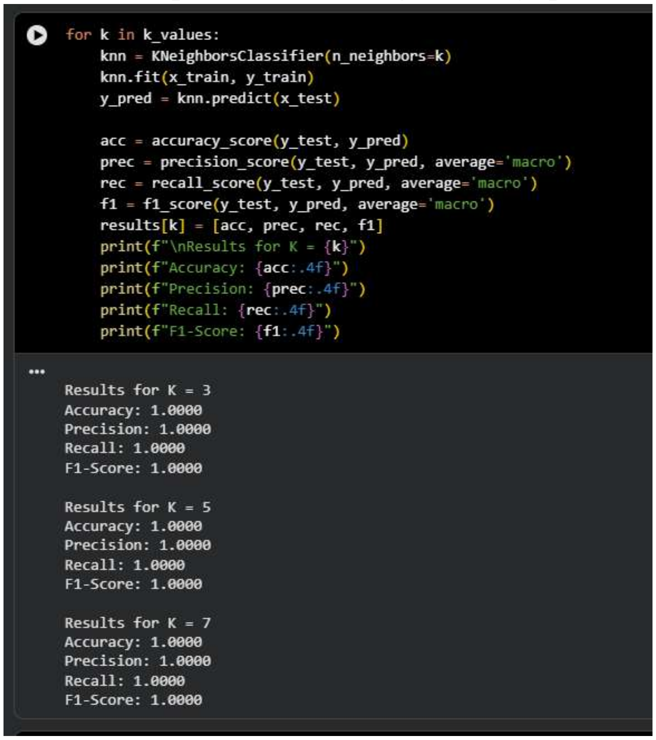
```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

4. Train a KNN classifier with different values of K (e.g., 3, 5, 7).



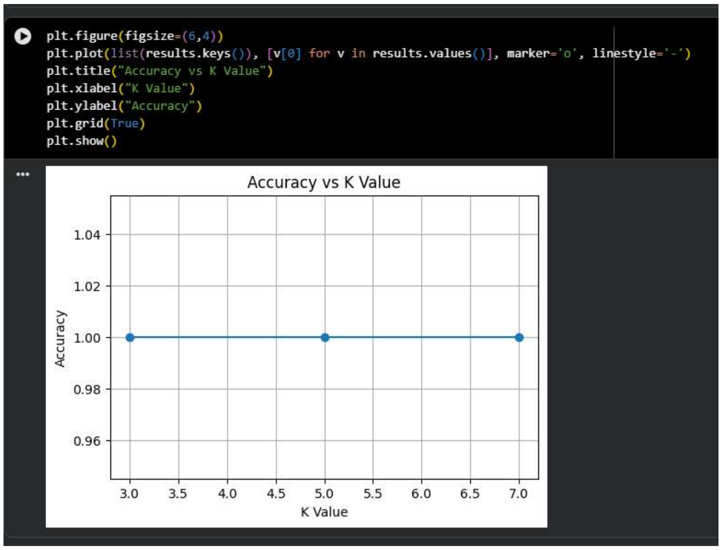
```
k_values = [3, 5, 7]
results = {}
```

5. Make predictions on the test set. AND
6. Evaluate performance using accuracy, precision, recall, and F1-score.



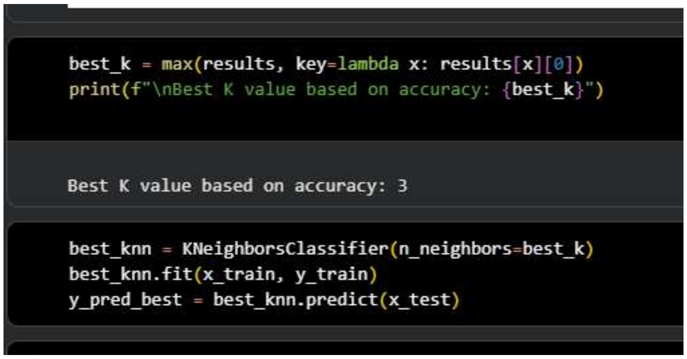
```
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)

    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')
    results[k] = [acc, prec, rec, f1]
    print(f"\nResults for K = {k}")
    print(f"Accuracy: {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall: {rec:.4f}")
    print(f"F1-Score: {f1:.4f}")
```

```

Results for K = 3
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000

Results for K = 5
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000

Results for K = 7
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000
```



```
plt.figure(figsize=(6,4))
plt.plot(list(results.keys()), [v[0] for v in results.values()], marker='o', linestyle='-')
plt.title("Accuracy vs K Value")
plt.xlabel("K Value")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
```

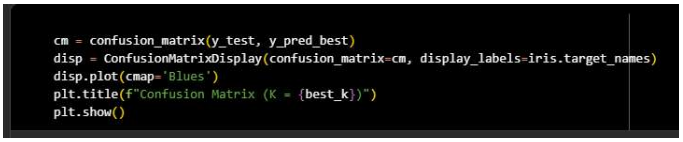
7. Test how accuracy changes with different values of K.



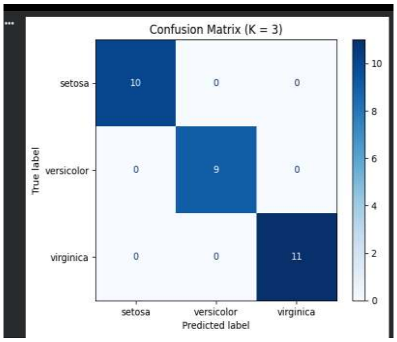
```
best_k = max(results, key=lambda x: results[x][0])
print(f"\nBest K value based on accuracy: {best_k}")
```

```
Best K value based on accuracy: 3
```

```
best_knn = KNeighborsClassifier(n_neighbors=best_k)
best_knn.fit(x_train, y_train)
y_pred_best = best_knn.predict(x_test)
```



```
cm = confusion_matrix(y_test, y_pred_best)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=iris.target_names)
disp.plot(cmap='Blues')
plt.title(f"Confusion Matrix (K = {best_k})")
plt.show()
```



Confusion Matrix (K = 3)
setosa
versicolor
virginica
True label
10
0
0
0
9
0
0
0
11
setosa
versicolor
virginica
Predicted label
10
8
6
4
2
0