Experiment - 4

Nikhil Kale
D15A/50

Aim - Implement K-Nearest Neighbors (KNN) and evaluate model performance

1. Dataset Source

Dataset used:
https://www.kaggle.com/datasets/uciml/iris

Title: Iris Dataset

2. Dataset Description

The Iris dataset is a classical multiclass classification dataset used in pattern recognition.

Number of records: 150
Number of features: 4 numerical features
Target variable: Species (3 classes)

Features:

- SepalLengthCm
- SepalWidthCm
- PetalLengthCm
- PetalWidthCm

Target Classes:

- Iris-setosa
- Iris-versicolor
- Iris-virginica

Characteristics:

- Balanced dataset (50 samples per class)
- No missing values
- Linearly separable for Setosa class
- Multiclass classification problem

3. Mathematical Formulation

K-Nearest Neighbors

KNN is a non-parametric algorithm.

For a test sample:

1. Compute distance to all training samples:

$d(x,xi)=\sum(xj-xij)2$d(x,xi)=∑(xj−xij)2

2. Select K nearest neighbors.
3. Assign majority class.

Prediction:

$y^=\text{Mode}(K \text{ nearest neighbors})$y^=Mode(K nearest neighbors)

4. Algorithm Limitations

- Computationally expensive for large datasets
- Sensitive to irrelevant features
- Sensitive to scaling
- Performance depends heavily on K value
- Suffers from curse of dimensionality

5. Methodology / Workflow

1. Dataset loading
2. Data exploration and visualization
3. Label encoding
4. Feature scaling
5. Train-test split
6. KNN training
7. Hyperparameter tuning (K selection)
8. Performance evaluation

Workflow:

Dataset → EDA → Preprocessing → Scaling → Split → KNN → Tune K → Evaluate

6. Performance Analysis

Evaluation Metrics:

- Accuracy
- Precision
- Recall
- F1-score
- Confusion Matrix
- ROC-AUC

Observations:

- Iris dataset gives high accuracy (>95%)
- Setosa class perfectly separable
- Optimal K improves generalization
- Distance weighting sometimes improves performance

7. Hyperparameter Tuning

Parameters tuned:

- n_neighbors (K value)
- weights (uniform/distance)
- metric (euclidean/manhattan)

Used GridSearchCV with 5-fold cross validation.

Effect:

- Small K → overfitting
- Large K → underfitting
- Optimal K balances bias-variance tradeoff

Code and Output -

[1] ✓ 3s

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

plt.style.use('seaborn-v0_8')
sns.set_palette("Set2")
```

[2] ✓ 0s

```
df = pd.read_csv("/content/Iris.csv")
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

[3] ✓ 0s

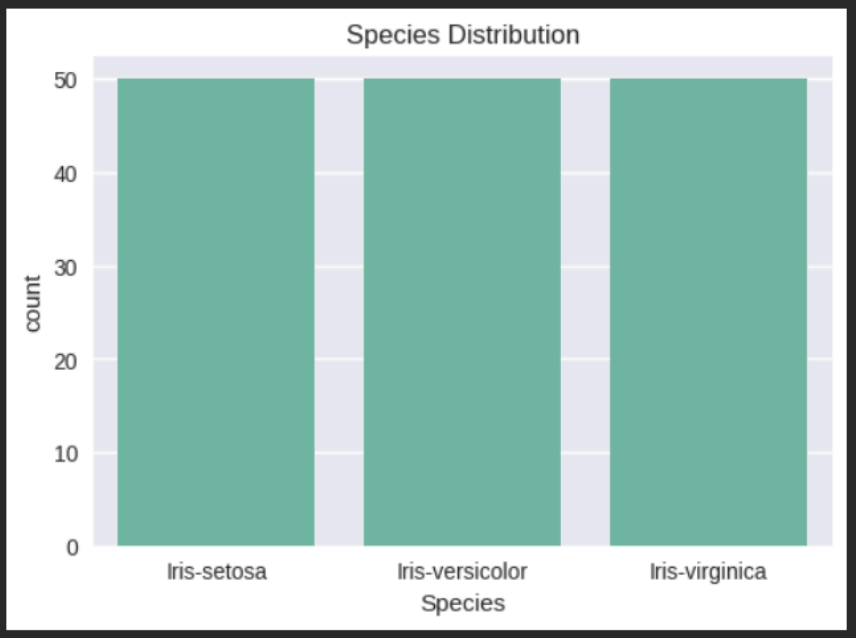
```
df.info()
df.describe()
df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

	0
Id	0
SepalLengthCm	0
SepalWidthCm	0
PetalLengthCm	0
PetalWidthCm	0
Species	0

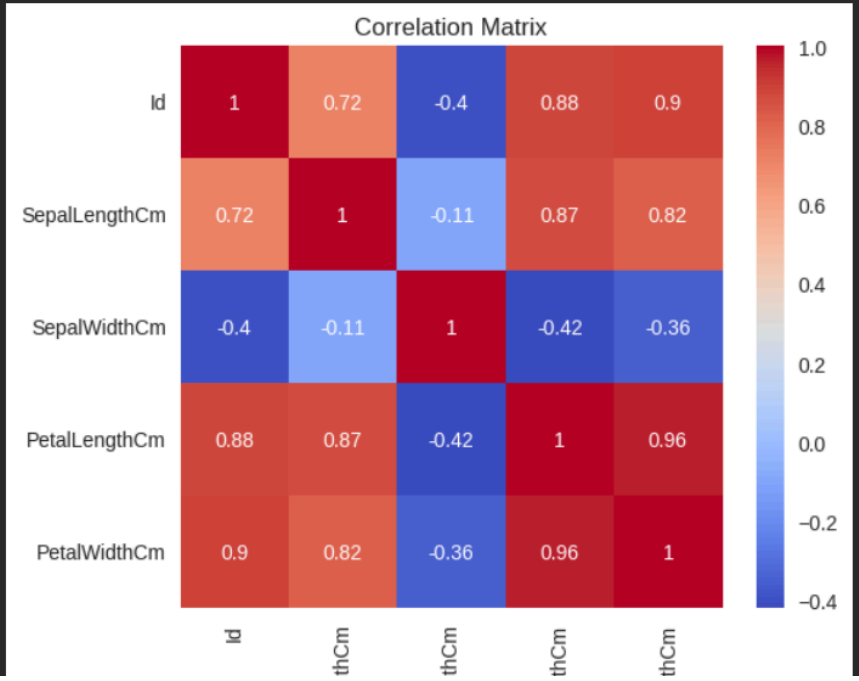
[4]
✓ 0s

```
plt.figure(figsize=(6,4))
sns.countplot(x='Species', data=df)
plt.title("Species Distribution")
plt.show()
```



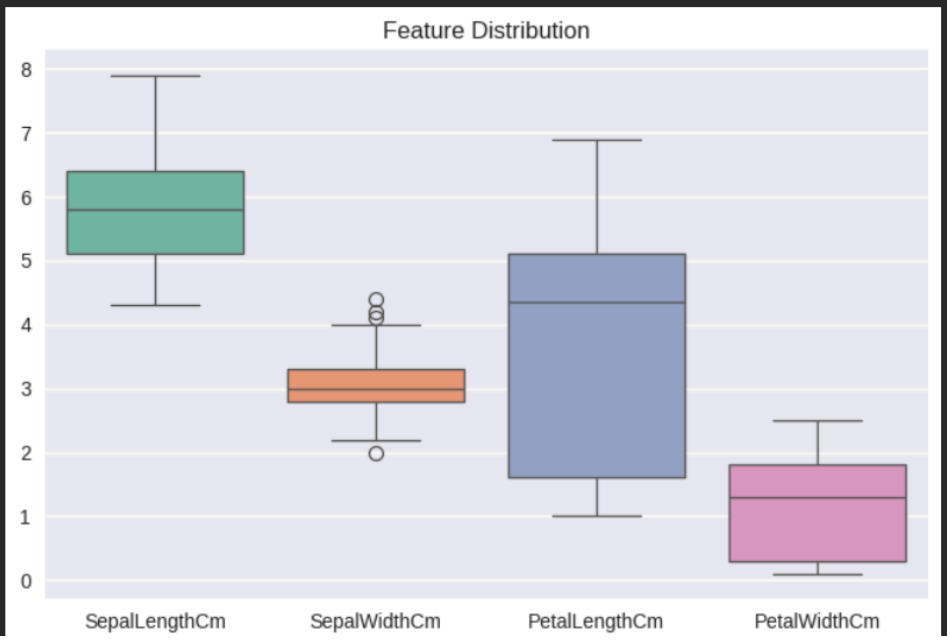
[7]

```
numeric_df = df.select_dtypes(include=np.number)
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```



[8]
✓ 0s

```
plt.figure(figsize=(8,5))
sns.boxplot(data=df.drop(["Id", "Species"], axis=1))
plt.title("Feature Distribution")
plt.show()
```



[9]
✓ 0s

```
le = LabelEncoder()
df['Species'] = le.fit_transform(df['Species'])
```

[10]
✓ 0s

```
X = df.drop(["Id", "Species"], axis=1)
y = df["Species"]
```

[11]
✓ 0s

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

[12]
✓ 0s

```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[13]
✓ 0s

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

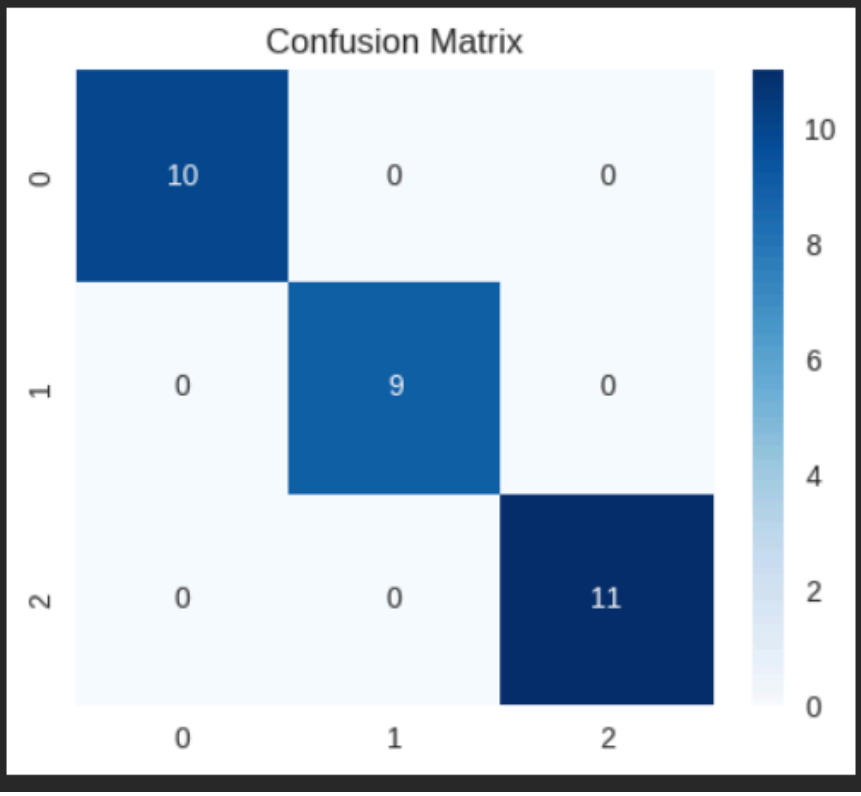
```
Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

[14]
✓ 1s

```
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```

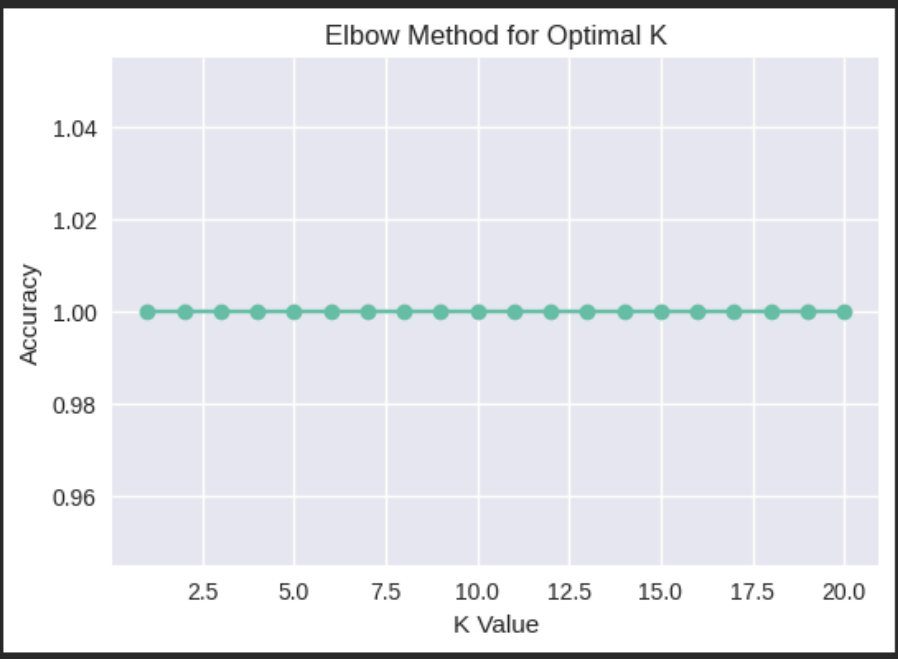


[15] ✓ 0s

```python
accuracy_scores = []

for k in range(1, 21):
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    accuracy_scores.append(accuracy_score(y_test, preds))

plt.figure(figsize=(6,4))
plt.plot(range(1,21), accuracy_scores, marker='o')
plt.xlabel("K Value")
plt.ylabel("Accuracy")
plt.title("Elbow Method for Optimal K")
plt.show()
```



[16] ✓ 2s

```python
param_grid = {
    'n_neighbors': list(range(1, 21)),
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)
grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)

best_knn = grid.best_estimator_
y_pred_tuned = best_knn.predict(X_test)

print("Tuned Accuracy:", accuracy_score(y_test, y_pred_tuned))
```

```
Best Parameters: {'metric': 'euclidean', 'n_neighbors': 9, 'weights': 'distance'}
Tuned Accuracy: 1.0
```

[17]
✓ 0s

```
y_test_bin = label_binarize(y_test, classes=[0,1,2])
y_score = best_knn.predict_proba(X_test)

for i in range(3):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'Class {i} (AUC = {roc_auc:.2f})')

plt.plot([0,1],[0,1],'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve (Multiclass)")
plt.legend()
plt.show()
```