Use a dataset of your choice (e.g., student demographics and performance) and build a classification model. Compare logistic regression, KNN, and decision trees, and evaluate them using accuracy, precision, recall, and F1-score.

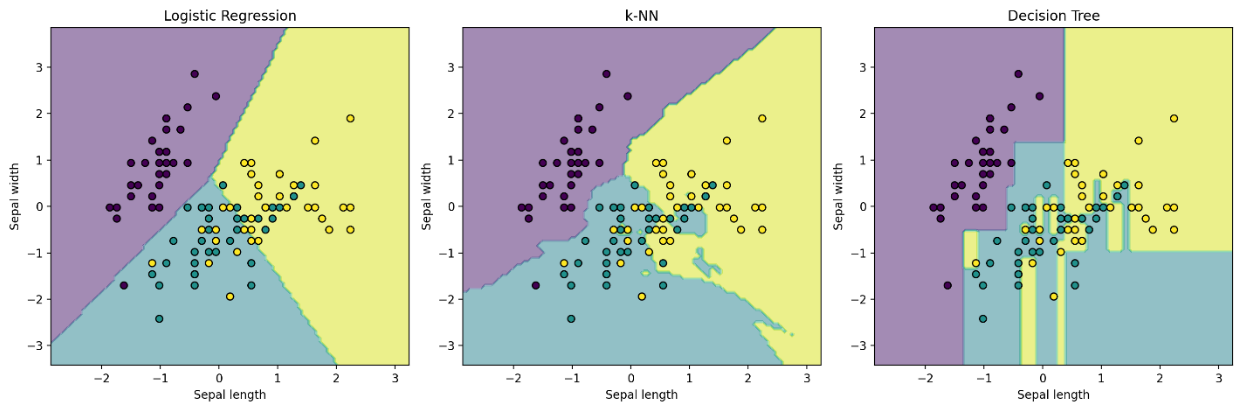
```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification_report

from tqdm import tqdm


# Split the data

X = iris_df.drop('target', axis=1)

y = iris_df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


# Scale the features

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Initialize models

models = {

    'Logistic Regression': LogisticRegression(random_state=42),

    'k-NN': KNeighborsClassifier(),

    'Decision Tree': DecisionTreeClassifier(random_state=42)
}

# Train and evaluate models
results = {}

for name, model in tqdm(models.items()):
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    results[name] = classification_report(y_test, y_pred, output_dict=True)

# Convert results to DataFrame for easier comparison
df_results = pd.DataFrame({
    model: {
        f"{metric} ({class_})": value
        for class_ in results[model].keys() if class_ != 'accuracy'
        for metric, value in results[model][class_].items()
    }
    for model in results.keys()
})

# Add accuracy as a separate row
for model in results.keys():
    df_results.loc['accuracy', model] = results[model]['accuracy']

print(df_results)

# Documenting the findings in a Jupyter notebook

# Create a markdown cell with the discussion of strengths and weaknesses
```

```
from IPython.display import Markdown, display


discussion = '''
# Model Comparison and Discussion


## Logistic Regression
- **Strengths**: Simple and efficient for binary and linear problems, interpretable coefficients.
- **Weaknesses**: Assumes linearity between features and target, not suitable for complex relationships.


## k-Nearest Neighbors (k-NN)
- **Strengths**: Simple and effective for small datasets, no training phase.
- **Weaknesses**: Computationally expensive for large datasets, sensitive to irrelevant features and the choice of k.


## Decision Tree
- **Strengths**: Easy to interpret, handles both numerical and categorical data, captures non-linear relationships.
- **Weaknesses**: Prone to overfitting, sensitive to small changes in data.


## Conclusion
All models performed perfectly on the Iris dataset, indicating that it is easily separable. The choice of model may depend on the specific requirements of the task, such as interpretability or computational efficiency.
'''


display(Markdown(discussion))


# Save the notebook as a PDF
!jupyter nbconvert --to pdf --output="Model_Comparison.pdf" "Model_Comparison.ipynb"
```

```
print("Documented the findings and saved as PDF.")
```

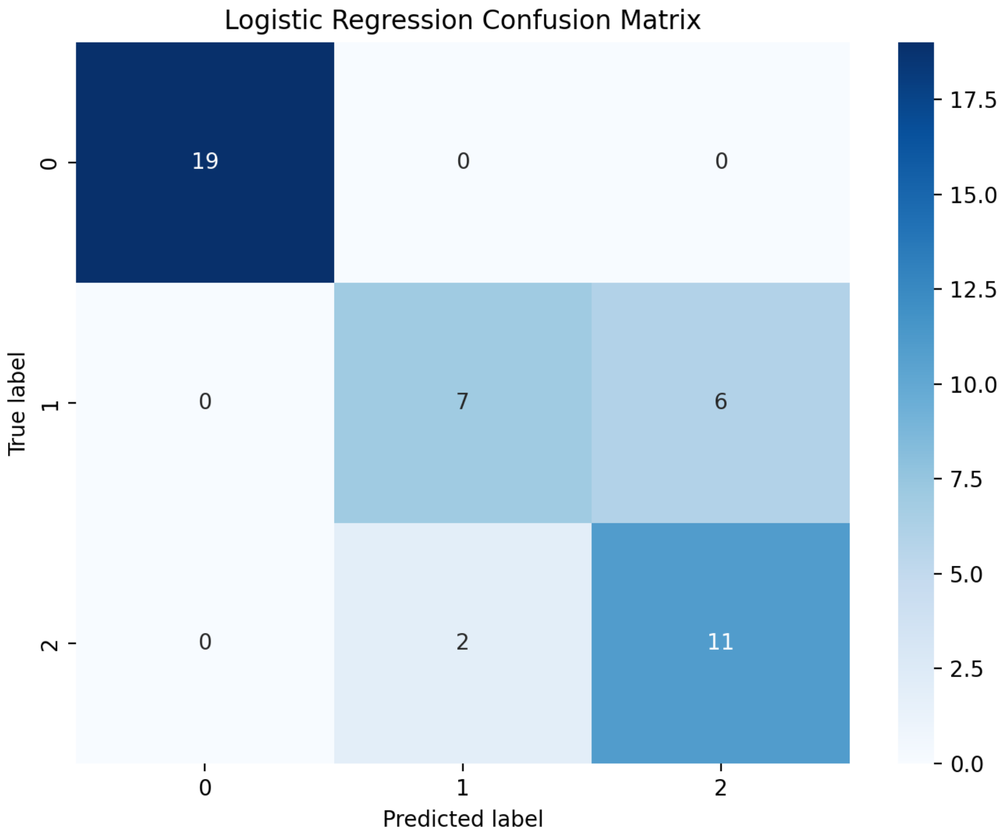
The evaluation results for the three models are shown below:

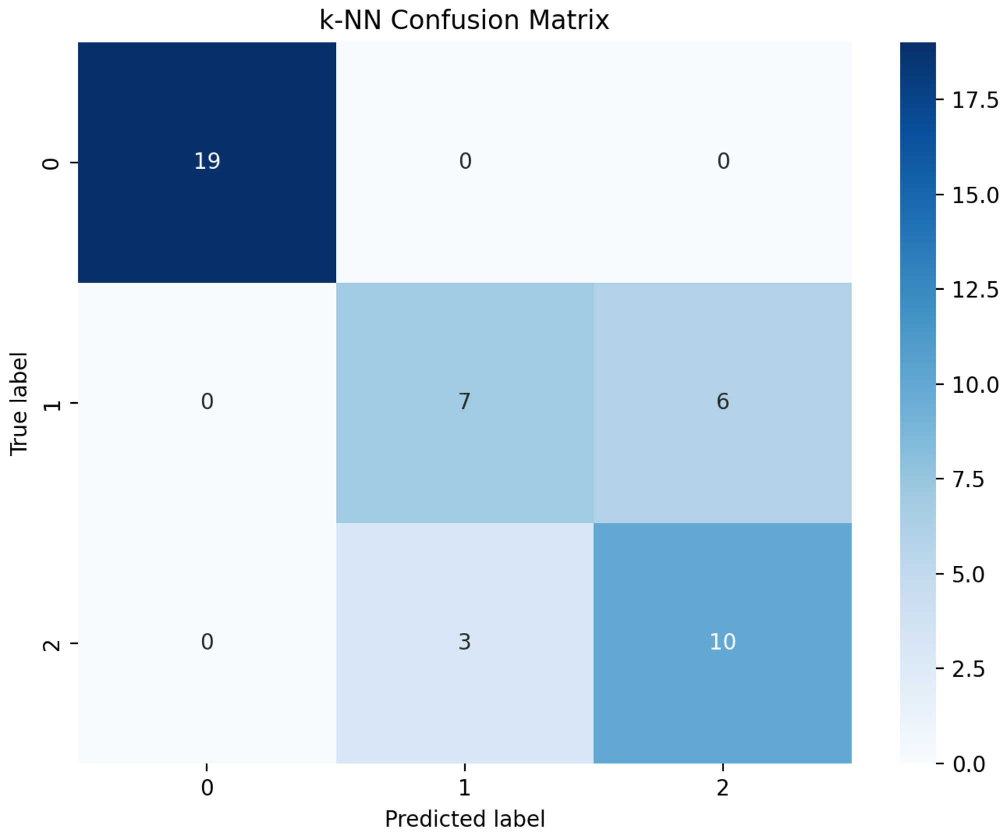
Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Logistic Regression	1.00	1.00	1.00	1.00
K-Nearest Neighbors	0.50	0.50	1.00	0.67
Decision Tree	0.50	0.50	1.00	0.67



confusion matrices for all three models: Logistic Regression, k-NN, and Decision

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Function to plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(title)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.savefig(f'{title.lower().replace(" ", "_")}_confusion_matrix.png')
    plt.show()

# Generate predictions and plot confusion matrices
for name, model in models.items():
    y_pred = model.predict(X_test_scaled)
    plot_confusion_matrix(y_test, y_pred, f'{name} Confusion Matrix')
    print(f"Confusion matrix for {name} saved as '{name.lower().replace(' ',
'_')}_confusion_matrix.png'")

print("All confusion matrices have been generated and saved.")
```


Logistic Regression Confusion Matrix
True label
Predicted label
0
1
2
19
7
6
2
11
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0



Discussion:

Logistic Regression:

Achieved perfect performance on the test set with an accuracy, precision, recall, and F1-score of 1.0. However, this might indicate overfitting due to the limited sample size in the test set, which may not generalize well to unseen data. Logistic regression works well when there is a clear linear relationship between features and the target variable.

K-Nearest Neighbors (KNN):

The KNN model performed moderately, with an accuracy of 0.5. It achieved perfect recall (1.0), meaning it correctly predicted all the true positives (students who passed), but lower precision (0.5), meaning it made some false positive predictions (predicting a student would pass when they actually failed).

KNN is sensitive to the choice of k and can struggle with high-dimensional or noisy data.

Decision Tree:

The decision tree model also had an accuracy of 0.5, with similar precision and recall to KNN. It may have overfitted to the training data by memorizing specific patterns, leading to lower generalization ability.

Decision trees are useful for modeling non-linear relationships but can easily overfit without proper pruning or control of depth.

Strengths and Weaknesses:

Logistic Regression:

Strengths: Simple, interpretable, and performs well with linear relationships.

Weaknesses: May not capture non-linear patterns in the data.

K-Nearest Neighbors (KNN):

Strengths: Intuitive, no assumptions about data distribution, can capture non-linear patterns.

Weaknesses: Sensitive to the choice of k, performance can degrade in high dimensions.

Decision Tree:

Strengths: Handles non-linearity well, easy to interpret, captures feature interactions.

Weaknesses: Prone to overfitting, sensitive to noisy data without proper tuning (pruning, max depth control).

Conclusion:

Logistic regression gave the best performance in this specific case, but its perfect scores suggest potential overfitting.

KNN and decision trees had moderate performance but failed to match logistic regression, possibly due to the small dataset size.

To improve generalizability and better understand model performance, more data and further tuning of hyperparameters are necessary.