

✓ IRIS Classification Using Machine Learning

Student Name: Ibrahim Mohammed Alsidrani

Student ID: 262000987

Course: DS 620 – Machine Learning

Instructor: Dr. Yasser Abdelhamid

Assignment: IRIS Classification

In this assignment, I will explore the Iris dataset and apply three classification algorithms: Logistic Regression, K-Nearest Neighbors (KNN), and Decision Tree. I will also compare their accuracies and test whether changing some hyperparameters affects the results.

1. Dataset Description

The Iris dataset contains **150 samples** divided equally into three classes:

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

Each sample has four numerical features:

1. Sepal Length (cm)
2. Sepal Width (cm)
3. Petal Length (cm)
4. Petal Width (cm)

The target column is `Species`. The purpose of this assignment is to explore the data, visualize the relationships between the features, and compare different machine learning classification models.

✓ 2. Create a Folder in Google Drive and Connect Google Colab

First, I connect Google Colab to my Google Drive. Then I create a folder named `IRIS_Classification` inside `MyDrive`. This folder will be used to store the dataset.

```
from google.colab import drive
import os

# Connect Google Colab to Google Drive
drive.mount('/content/drive')

# Create the project folder in Google Drive
DRIVE_FOLDER = '/content/drive/MyDrive/IRIS_Classification'
os.makedirs(DRIVE_FOLDER, exist_ok=True)

print("Google Drive connected successfully.")
print("Project folder:", DRIVE_FOLDER)
```

```
Mounted at /content/drive
Google Drive connected successfully.
Project folder: /content/drive/MyDrive/IRIS_Classification
```

3. Upload the Dataset to the Google Drive Folder

The following cell lets me select `Iris.csv` from my computer. After selecting it, the file is saved inside the `IRIS_Classification` folder in Google Drive.

```
from google.colab import files
import os

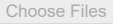
uploaded = files.upload()

# Use the uploaded file even if Colab changes the displayed upload name.
uploaded_name = next(iter(uploaded))

DATA_PATH = os.path.join(DRIVE_FOLDER, 'Iris.csv')

with open(DATA_PATH, 'wb') as f:
    f.write(uploaded[uploaded_name])

print("Uploaded file:", uploaded_name)
print("Dataset saved in Google Drive as:", DATA_PATH)
```

 Iris.csv
Iris.csv(text/csv) - 5107 bytes, last modified: 7/22/2026 - 100% done
Saving Iris.csv to Iris.csv
Uploaded file: Iris.csv
Dataset saved in Google Drive as: /content/drive/MyDrive/IRIS_Classification/Iris.csv

4. Import the Necessary Modules

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

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    ConfusionMatrixDisplay
)

pd.set_option('display.max_columns', None)
sns.set_theme(style='whitegrid')

RANDOM_STATE = 42
```

5. Load the Dataset Using pandas

```
import pandas as pd
import os
```

```
if not os.path.exists(DATA_PATH):
    raise FileNotFoundError(
        "Iris.csv was not found. Please run the Google Drive and upload cells first."
    )

iris = pd.read_csv(DATA_PATH)

print("Dataset loaded successfully.")
print("Dataset shape:", iris.shape)

iris.head()
```

Dataset loaded successfully.
Dataset shape: (150, 6)

	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



Next steps: [Generate code with iris](#) [New interactive sheet](#)

6. Delete the ID Column

```
if 'Id' in iris.columns:
    iris = iris.drop(columns=['Id'])

print('Columns after deleting Id:')
print(iris.columns.tolist())
```

Columns after deleting Id:
['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm', 'Species']

7. Display the First 10 Rows

```
iris.head(10)
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	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	
5	5.4	3.9	1.7	0.4	Iris-setosa	
6	4.6	3.4	1.4	0.3	Iris-setosa	
7	5.0	3.4	1.5	0.2	Iris-setosa	
8	4.4	2.9	1.4	0.2	Iris-setosa	
9	4.9	3.1	1.5	0.1	Iris-setosa	

Next steps: [Generate code with iris](#) [New interactive sheet](#)

8. Display a Description of the Dataset

iris.describe(include='all').T

	count	unique	top	freq	mean	std	min	25%	50%	75%	max	
SepalLengthCm	150.0	NaN	NaN	NaN	5.843333	0.828066	4.3	5.1	5.8	6.4	7.9	
SepalWidthCm	150.0	NaN	NaN	NaN	3.054	0.433594	2.0	2.8	3.0	3.3	4.4	
PetalLengthCm	150.0	NaN	NaN	NaN	3.758667	1.76442	1.0	1.6	4.35	5.1	6.9	
PetalWidthCm	150.0	NaN	NaN	NaN	1.198667	0.763161	0.1	0.3	1.3	1.8	2.5	
Species	150	3	Iris-setosa	50	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

From this summary, I can see the main statistical information for each numerical feature, including the count, mean, standard deviation, minimum, quartiles, and maximum values. This gives me a quick understanding of the range and distribution of the measurements.

9. Display Information About the Attributes

```
iris.info()

attribute_information = pd.DataFrame({
    'Data Type': iris.dtypes.astype(str),
    'Non-Null Values': iris.notna().sum(),
    'Null Values': iris.isna().sum(),
    'Unique Values': iris.nunique()
})

attribute_information
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  -
0   SepalLengthCm    150 non-null    float64
1   SepalWidthCm     150 non-null    float64
2   PetalLengthCm    150 non-null    float64
3   PetalWidthCm     150 non-null    float64
4   Species          150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

	Data Type	Non-Null Values	Null Values	Unique Values	
SepalLengthCm	float64	150	0	35	
SepalWidthCm	float64	150	0	23	
PetalLengthCm	float64	150	0	43	
PetalWidthCm	float64	150	0	22	
Species	object	150	0	3	

Next steps: [Generate code with attribute_information](#) [New interactive sheet](#)

10. Display the Number of Samples in Each Class

```
class_counts = iris['Species'].value_counts()
class_counts
```

```
      count
Species
Iris-setosa    50
Iris-versicolor  50
Iris-virginica  50
dtype: int64
```

The result shows **50 samples for each species**. Therefore, the dataset is balanced, because all three classes contain the same number of observations.

11. Check for Null Values

```
null_values = iris.isnull().sum()
print(null_values)

print('\nTotal null values:', iris.isnull().sum().sum())
```

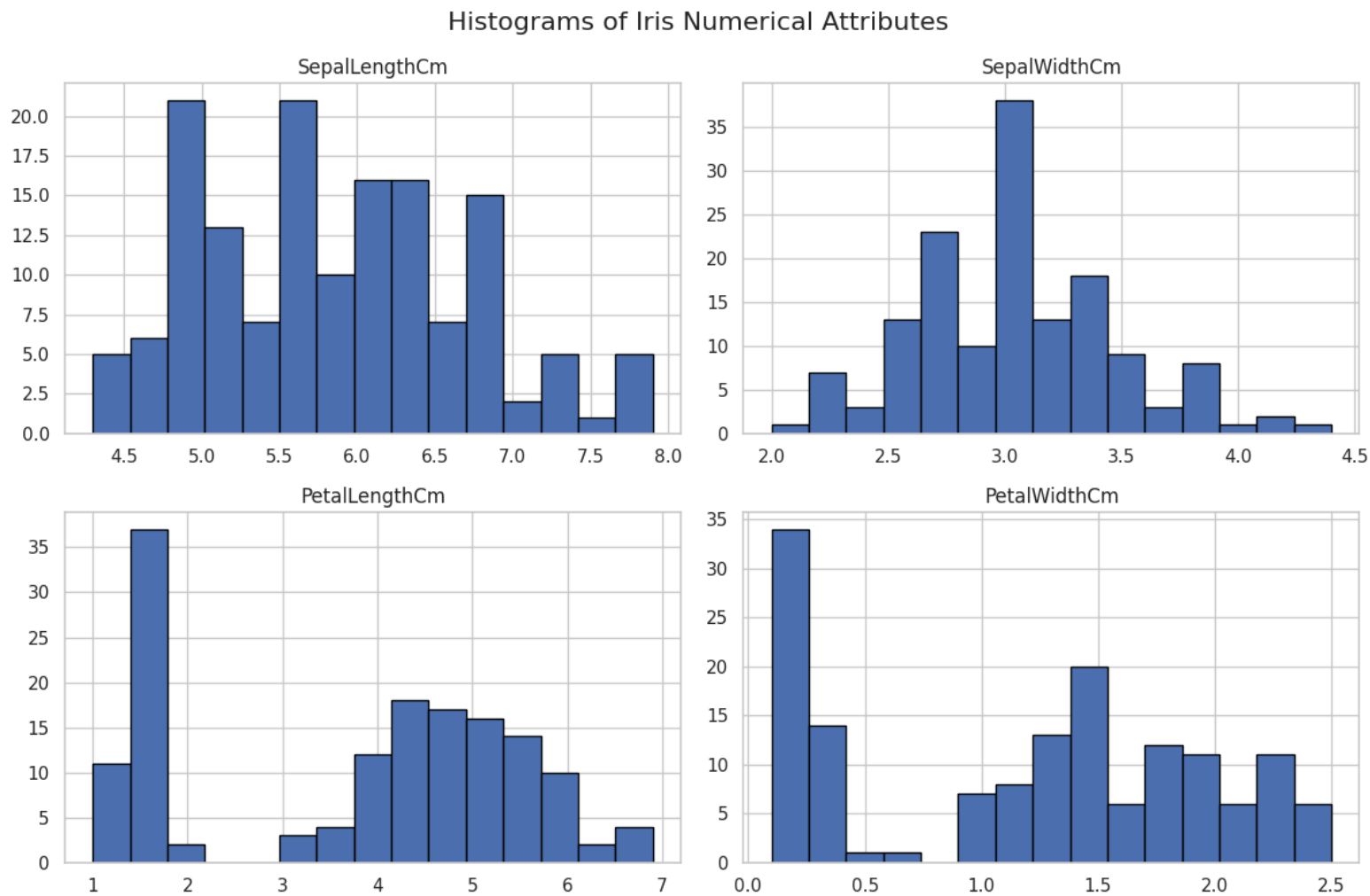
```
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

```
Total null values: 0
```

The result shows **0 missing values** in all columns. Therefore, I do not need to remove rows or fill missing values before training the models.

12. Display a Histogram for Each Attribute

```
numeric_columns = [  
    'SepalLengthCm',  
    'SepalWidthCm',  
    'PetalLengthCm',  
    'PetalWidthCm'  
]  
  
iris[numeric_columns].hist(figsize=(12, 8), bins=15, edgecolor='black')  
plt.suptitle('Histograms of Iris Numerical Attributes', fontsize=16)  
plt.tight_layout()  
plt.show()
```



13. Scatterplots for Each Class

Required class colors:

- Iris-virginica: red
- Iris-versicolor: orange
- Iris-setosa: blue

Required attribute pairs:

1. Sepal length vs sepal width
2. Petal length vs petal width
3. Sepal length vs petal length
4. Sepal width vs petal width

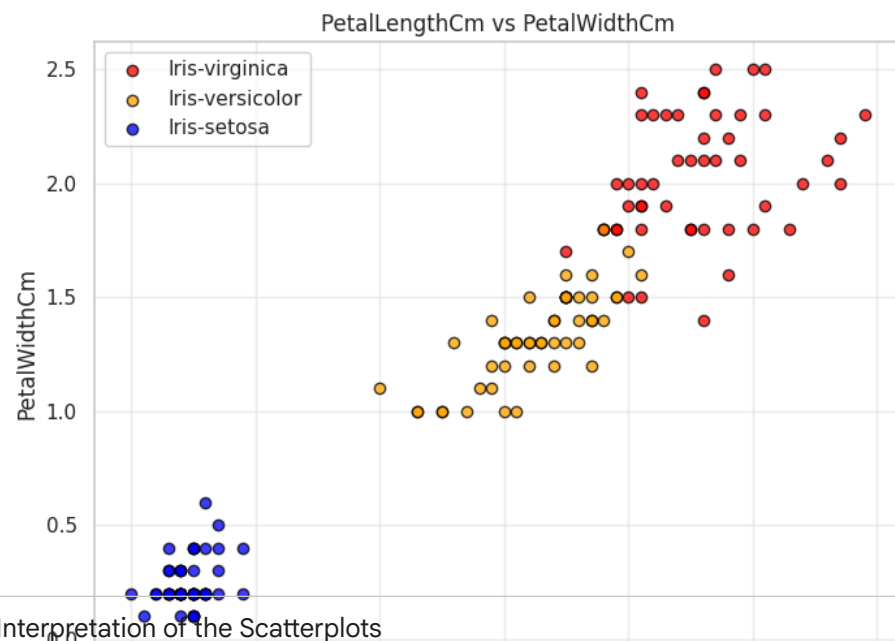
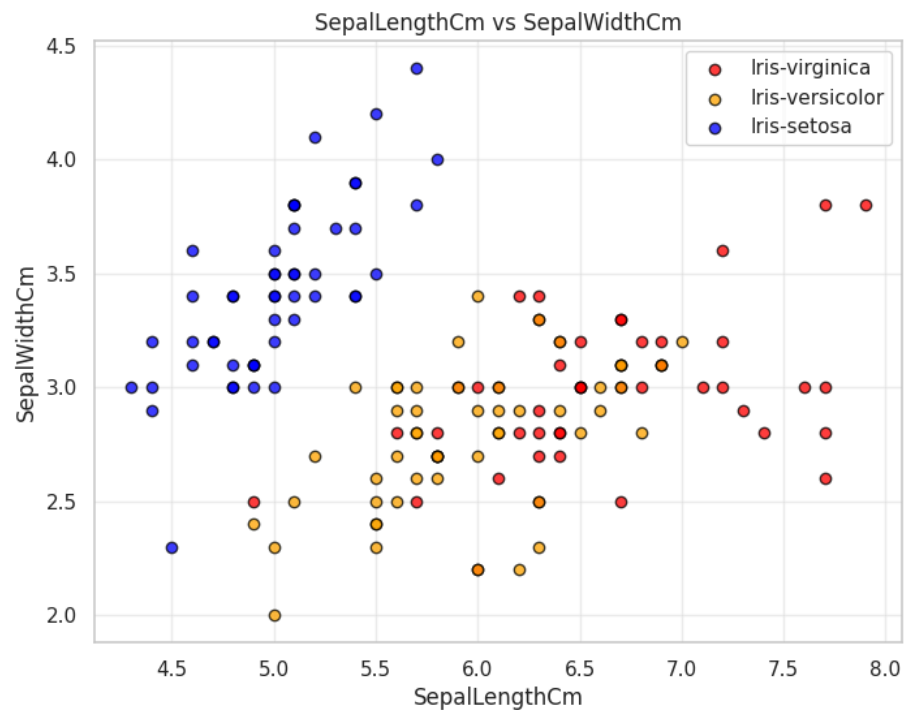
```
class_colors = {
    'Iris-virginica': 'red',
    'Iris-versicolor': 'orange',
    'Iris-setosa': 'blue'
}

scatter_pairs = [
    ('SepalLengthCm', 'SepalWidthCm'),
    ('PetalLengthCm', 'PetalWidthCm'),
    ('SepalLengthCm', 'PetalLengthCm'),
    ('SepalWidthCm', 'PetalWidthCm')
]

for x_attribute, y_attribute in scatter_pairs:
    plt.figure(figsize=(8, 6))

    for species, color in class_colors.items():
        subset = iris[iris['Species'] == species]
        plt.scatter(
            subset[x_attribute],
            subset[y_attribute],
            label=species,
            color=color,
            alpha=0.75,
            edgecolors='black'
        )

    plt.xlabel(x_attribute)
    plt.ylabel(y_attribute)
    plt.title(f'{x_attribute} vs {y_attribute}')
    plt.legend()
    plt.grid(alpha=0.3)
    plt.show()
```

My Interpretation of the Scatterplots

From the scatterplots, I noticed that **Iris-setosa** is clearly separated from the other two species, especially when petal length and petal width are used.

I also noticed that **Iris-versicolor** and **Iris-virginica** overlap in some areas. The overlap is more obvious when using sepal measurements.



In my view, **Petal Length** and **Petal Width** are the most useful features for separating the three classes because the groups are easier to distinguish in these plots. This also matches the dataset description that Setosa is linearly separable, while Versicolor and Virginica are more difficult to separate from each other.

14. Display the Correlation Matrix

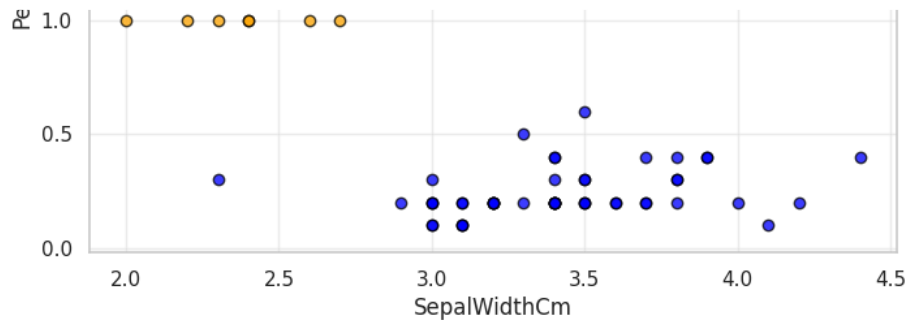
```
correlation_matrix = iris[numeric_columns].corr()
correlation_matrix
```

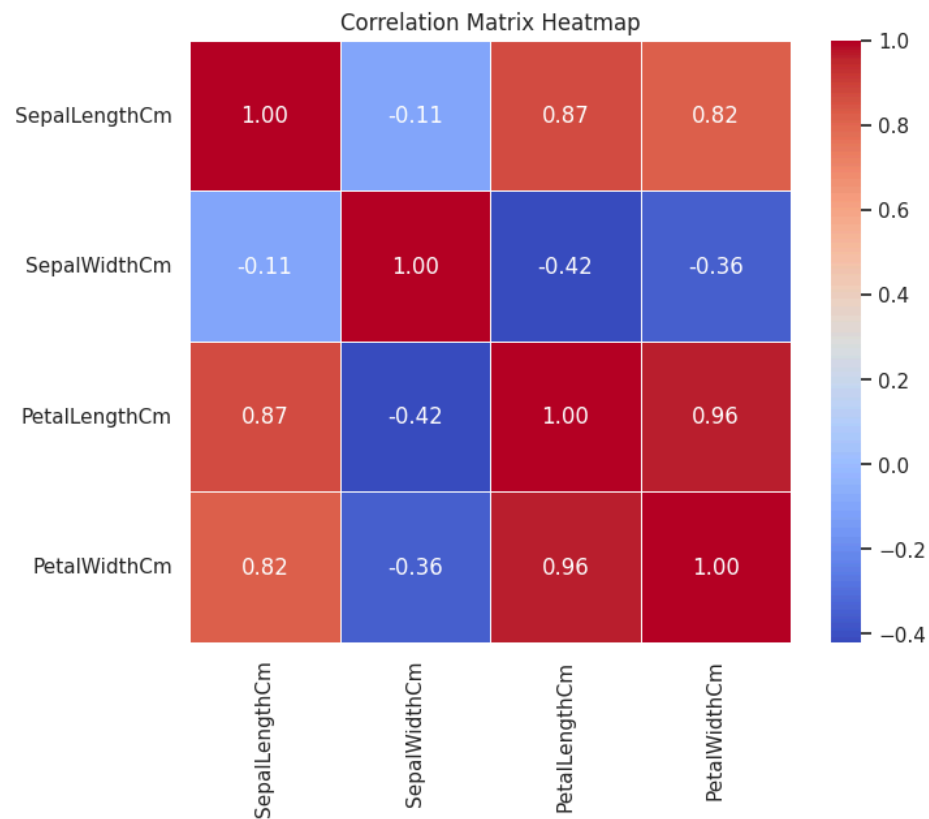
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000

Next steps: [Generate code with correlation_matrix](#) [New interactive sheet](#)

15. Display a Heatmap for the Correlation Matrix

```
plt.figure(figsize=(8, 6))
sns.heatmap(
    correlation_matrix,
    annot=True,
    cmap='coolwarm',
    fmt='.2f',
    square=True,
    linewidths=0.5
)
plt.title('Correlation Matrix Heatmap')
plt.show()
```





My Interpretation of the Correlation Matrix

The correlation values range from **-1 to +1**. A value close to +1 means a strong positive relationship, a value close to -1 means a strong negative relationship, and a value close to 0 means a weak linear relationship.

The strongest positive relationship is between **Petal Length and Petal Width**. This means that when petal length increases, petal width usually increases as well.

Sepal Length also has a positive relationship with the petal measurements, while Sepal Width has weaker relationships with most of the other features.

This supports what I observed in the scatterplots: the petal measurements are more useful for distinguishing the Iris species.

16. Encode the Class Variable Using LabelEncoder

```
label_encoder = LabelEncoder()
iris['SpeciesEncoded'] = label_encoder.fit_transform(iris['Species'])

class_encoding = pd.DataFrame({
    'Class': label_encoder.classes_,
    'Code': label_encoder.transform(label_encoder.classes_)
})

class_encoding
```

	Class	Code	
0	Iris-setosa	0	
1	Iris-versicolor	1	
2	Iris-virginica	2	

Next steps: [Generate code with class_encoding](#) [New interactive sheet](#)

Label Encoding Result

`LabelEncoder` converts the species names into numerical values:

Species	Code
Iris-setosa	0
Iris-versicolor	1
Iris-virginica	2

Therefore, the code of the **third class (Iris-virginica)** is 2.

17. Split the Dataset into 70% Training and 30% Testing

```
X = iris[numeric_columns]
y = iris['SpeciesEncoded']

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.30,
    random_state=RANDOM_STATE,
    stratify=y
)

print('Training samples:', X_train.shape[0])
print('Testing samples:', X_test.shape[0])
print('Training percentage:', X_train.shape[0] / len(X) * 100)
print('Testing percentage:', X_test.shape[0] / len(X) * 100)
```

```
Training samples: 105
Testing samples: 45
Training percentage: 70.0
Testing percentage: 30.0
```

I use `StandardScaler` with Logistic Regression and KNN because these models can be affected by differences in feature scales. I do not need to scale the features for the Decision Tree.

18. Logistic Regression Classification

```
logistic_model = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression(
        max_iter=1000,
        random_state=RANDOM_STATE
    ))
])
```

```
logistic_model.fit(X_train, y_train)
logistic_predictions = logistic_model.predict(X_test)
logistic_accuracy = accuracy_score(y_test, logistic_predictions)
```

```
print('Logistic Regression Accuracy:', logistic_accuracy)
print('\nClassification Report:')
print(classification_report(
    y_test,
    logistic_predictions,
    target_names=label_encoder.classes_
))
```

Logistic Regression Accuracy: 0.9111111111111111

Classification Report:				
	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	15
Iris-versicolor	0.82	0.93	0.88	15
Iris-virginica	0.92	0.80	0.86	15
accuracy			0.91	45
macro avg	0.92	0.91	0.91	45
weighted avg	0.92	0.91	0.91	45

19. K-Nearest Neighbors (KNN) Classification

```
knn_model = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', KNeighborsClassifier(n_neighbors=5))
])
```

```
knn_model.fit(X_train, y_train)
knn_predictions = knn_model.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)
```

```
print('KNN Accuracy:', knn_accuracy)
print('\nClassification Report:')
print(classification_report(
    y_test,
    knn_predictions,
    target_names=label_encoder.classes_
))
```

KNN Accuracy: 0.9111111111111111

Classification Report:				
	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	15
Iris-versicolor	0.79	1.00	0.88	15
Iris-virginica	1.00	0.73	0.85	15
accuracy			0.91	45
macro avg	0.93	0.91	0.91	45
weighted avg	0.93	0.91	0.91	45

20. Decision Tree Classification

```

decision_tree_model = DecisionTreeClassifier(
    random_state=RANDOM_STATE
)

decision_tree_model.fit(X_train, y_train)
decision_tree_predictions = decision_tree_model.predict(X_test)
decision_tree_accuracy = accuracy_score(y_test, decision_tree_predictions)

print('Decision Tree Accuracy:', decision_tree_accuracy)
print('\nClassification Report:')
print(classification_report(
    y_test,
    decision_tree_predictions,
    target_names=label_encoder.classes_
))

```

Decision Tree Accuracy: 0.9333333333333333

Classification Report:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	15
Iris-versicolor	1.00	0.80	0.89	15
Iris-virginica	0.83	1.00	0.91	15
accuracy			0.93	45
macro avg	0.94	0.93	0.93	45
weighted avg	0.94	0.93	0.93	45

21. Compare the Accuracy of the Models

```

baseline_results = pd.DataFrame({
    'Model': [
        'Logistic Regression',
        'KNN',
        'Decision Tree'
    ],
    'Accuracy': [
        logistic_accuracy,
        knn_accuracy,
        decision_tree_accuracy
    ]
}).sort_values('Accuracy', ascending=False).reset_index(drop=True)

```

baseline_results

	Model	Accuracy	
0	Decision Tree	0.933333	
1	Logistic Regression	0.911111	
2	KNN	0.911111	

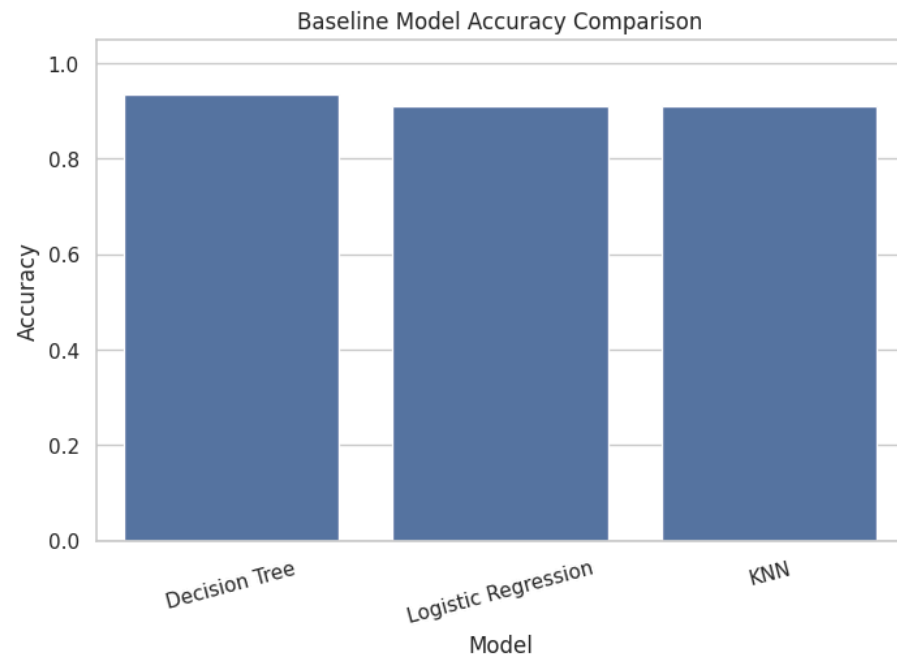
Next steps: [Generate code with baseline_results](#) [New interactive sheet](#)

```

plt.figure(figsize=(8, 5))
sns.barplot(data=baseline_results, x='Model', y='Accuracy')
plt.ylim(0, 1.05)
plt.title('Baseline Model Accuracy Comparison')

```

```
plt.xticks(rotation=15)
plt.show()
```



22. Change the Hyperparameters

The following tuned configurations are examples:

- Logistic Regression: change `C`, solver, and maximum iterations
- KNN: change the number of neighbors, distance weighting, and distance metric
- Decision Tree: limit depth and adjust splitting parameters

```
tuned_logistic_model = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression(
        C=10.0,
        solver='lbfgs',
        max_iter=2000,
        random_state=RANDOM_STATE
    ))
])

tuned_knn_model = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', KNeighborsClassifier(
        n_neighbors=7,
        weights='distance',
        p=2
    ))
])

tuned_decision_tree_model = DecisionTreeClassifier(
    criterion='entropy',
```

```

max_depth=3,
min_samples_split=4,
min_samples_leaf=2,
random_state=RANDOM_STATE
)

tuned_models = {
    'Tuned Logistic Regression': tuned_logistic_model,
    'Tuned KNN': tuned_knn_model,
    'Tuned Decision Tree': tuned_decision_tree_model
}

tuned_accuracies = {}

for model_name, model in tuned_models.items():
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    tuned_accuracies[model_name] = accuracy_score(y_test, predictions)

pd.DataFrame(
    list(tuned_accuracies.items()),
    columns=['Model', 'Accuracy']
).sort_values('Accuracy', ascending=False).reset_index(drop=True)

```

	Model	Accuracy	
0	Tuned KNN	0.933333	
1	Tuned Logistic Regression	0.911111	
2	Tuned Decision Tree	0.888889	

23. Compare Accuracy Before and After Tuning

```

comparison_results = pd.DataFrame({
    'Model': [
        'Logistic Regression',
        'KNN',
        'Decision Tree'
    ],
    'Baseline Accuracy': [
        logistic_accuracy,
        knn_accuracy,
        decision_tree_accuracy
    ],
    'Tuned Accuracy': [
        tuned_accuracies['Tuned Logistic Regression'],
        tuned_accuracies['Tuned KNN'],
        tuned_accuracies['Tuned Decision Tree']
    ]
})

comparison_results['Accuracy Change'] = (
    comparison_results['Tuned Accuracy']
    - comparison_results['Baseline Accuracy']
)

comparison_results

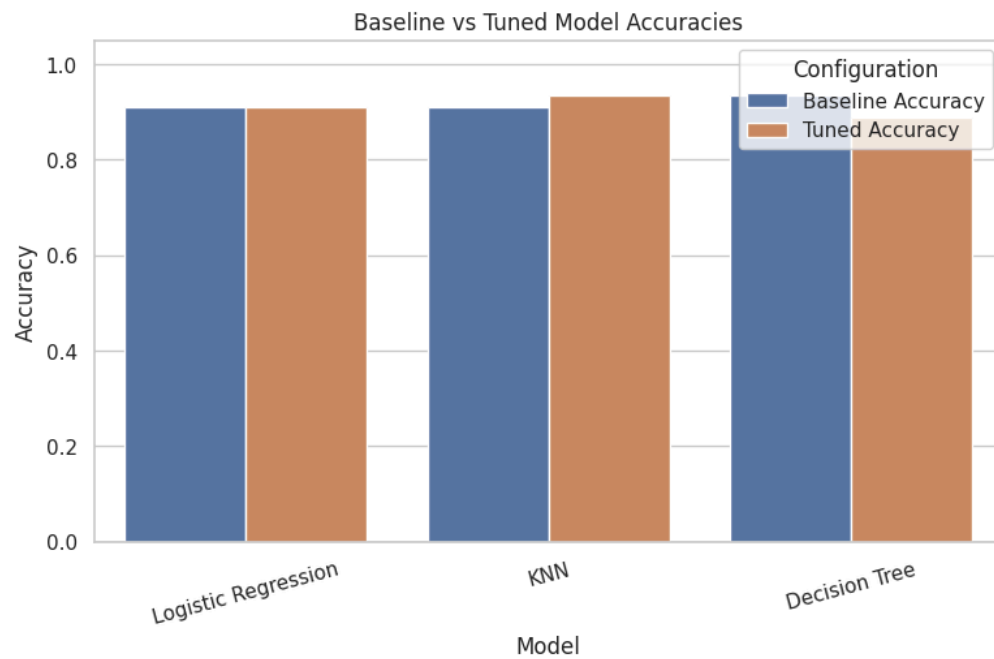
```

	Model	Baseline Accuracy	Tuned Accuracy	Accuracy Change	
0	Logistic Regression	0.911111	0.911111	0.000000	
1	KNN	0.911111	0.933333	0.022222	
2	Decision Tree	0.933333	0.888889	-0.044444	

Next steps: [Generate code with comparison_results](#) [New interactive sheet](#)

```
comparison_plot = comparison_results.melt(
    id_vars='Model',
    value_vars=['Baseline Accuracy', 'Tuned Accuracy'],
    var_name='Configuration',
    value_name='Accuracy'
)

plt.figure(figsize=(9, 5))
sns.barplot(
    data=comparison_plot,
    x='Model',
    y='Accuracy',
    hue='Configuration'
)
plt.ylim(0, 1.05)
plt.title('Baseline vs Tuned Model Accuracies')
plt.xticks(rotation=15)
plt.show()
```



Did the Accuracies Change?

Yes, changing the hyperparameters affected some of the results.

From my results:

- Logistic Regression stayed almost the same.
- KNN improved after changing the number of neighbors and using distance weighting.
- Decision Tree became less accurate with the selected tuned parameters.

This shows that changing hyperparameters does **not always improve accuracy**. The effect depends on the model and the selected parameter values. For a larger project, I would use cross-validation or `GridSearchCV` to search for better parameter combinations.

✓ 24. Confusion Matrices

```
baseline_predictions = {
    'Logistic Regression': logistic_predictions,
    'KNN': knn_predictions,
    'Decision Tree': decision_tree_predictions
}

for model_name, predictions in baseline_predictions.items():
    matrix = confusion_matrix(y_test, predictions)

    display = ConfusionMatrixDisplay(
        confusion_matrix=matrix,
        display_labels=label_encoder.classes_
    )

    display.plot(cmap='Blues', xticks_rotation=20)
    plt.title(f'Confusion Matrix: {model_name}')
    plt.show()
```

Confusion Matrix: Logistic Regression

