IRIS FLOWER CLASSIFICATION

Objective:

The main objective of this case study is to build and evaluate machine learning models to classify Iris flowers into one of three species: setosa, versicolor, or virginica based on their sepal and petal measurements. This is a classification problem that will involve data preprocessing, model training, evaluation, and prediction

Data Dictionary:

The Iris dataset contains the following features

- **sepal_length:** The length of the sepal (in cm)
- **sepal_width:** The width of the sepal (in cm)
- **petal_length:** The length of the petal (in cm)
- **petal_width:** The width of the petal (in cm)
- **species:** The class label representing the species of the Iris flower, with three possible
- **categories:** setosa, versicolor, and virginica

Key Steps:

- **Load Dataset:** Import and inspect the data for readiness.
- **Check Missing Values**: Identify and handle any missing data.
- **Visualize Target**: Plot class distribution for target balance.
- **Feature-Target Split**: Separate features (X) and target (y), then split into train/test sets.
- **Model Training & Evaluation**: Train models (e.g., Logistic Regression, Random Forest), evaluate performance.
- **Correlation Analysis**: Examine feature relationships with a heatmap.
- **Prediction**: Use the model to classify new flower data.

Packages:

- **pandas (pd)**: Data manipulation and analysis; works with DataFrame for tabular data.
- **numpy (np)**: Numerical computing; handles arrays and mathematical functions.
- **train_test_split**: Splits data into training and test sets.
- **Logistic Regression**: Performs logistic regression for classification tasks.
- **RandomForestClassifier**: An ensemble method for classification using multiple decision trees.
- **accuracy_score**: Calculates the accuracy of a classification model.
- **confusion_matrix**: Evaluates classification performance by showing true/false positives/negatives.
- **classification_report**: Provides precision, recall, and F1-score for each class.

- **seaborn (sns)**: Data visualization library for creating attractive statistical plots.
- **matplotlib (plt)**: Library for creating static, animated, and interactive plots.

Step 1: Load the Dataset and Libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Step 1: Load the Dataset
df = pd.read_csv('iris_is.csv', encoding='ISO-8859-1')  # Load the dataset
print("Dataset loaded successfully!:",df)
print("\nFirst 5 rows of the dataset:\n", df.head())  # Display the first 5
rows
print("\nInfo about the dataset:\n", df.info())  # Dataset structure and types
print("\nStatistical description of the dataset:\n", df.describe())  # Summary
statistics
```

Output

Dataset loaded successfully!:

	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
..	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

[150 rows x 5 columns]

First 5 rows of the dataset:

	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

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 150 entries, 0 to 149

Data columns (total 5 columns):

#	**Column**	**Non-Null Count**	**Dtype**
---	------	--------------	-----
0	sepal_length	150 non-null	float64
1	sepal_width	150 non-null	float64
2	petal_length	150 non-null	float64
3	petal_width	150 non-null	float64
4	species	150 non-null	object

dtypes: float64(4), object(1)

memory usage: 6.0+ KB

Info about the dataset: None

Statistical description of the dataset:

	sepal_length	**sepal_width**	**petal_length**	**petal_width**
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667

std	0.828066	0.433594	1 .764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Step 2: Check for Missing Values and Unique values

```
print("\nMissing values in each column:\n", df.isnull().sum())  # Identify
missing values
print("\nUnique species in the dataset:", df['species'].unique())  # Unique
values in the target column
```

Output:

Missing values in each column:

sepal_length 0

sepal_width 0

petal_length 0

petal_width 0

species 0

dtype: int64

Unique species in the dataset: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']

Step 3: Feature and Target Variables Split:

```
Splitting Features and Target Variables
X = df.drop(columns=['species'])  # Features (independent variables)
y = df['species']  # Target variable (dependent variable)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
print("\nData split successfully!")
print("Training features shape:", X_train.shape)  # Dimensions of training data
print("Testing features shape:", X_test.shape)  # Dimensions of testing data
```

Output:

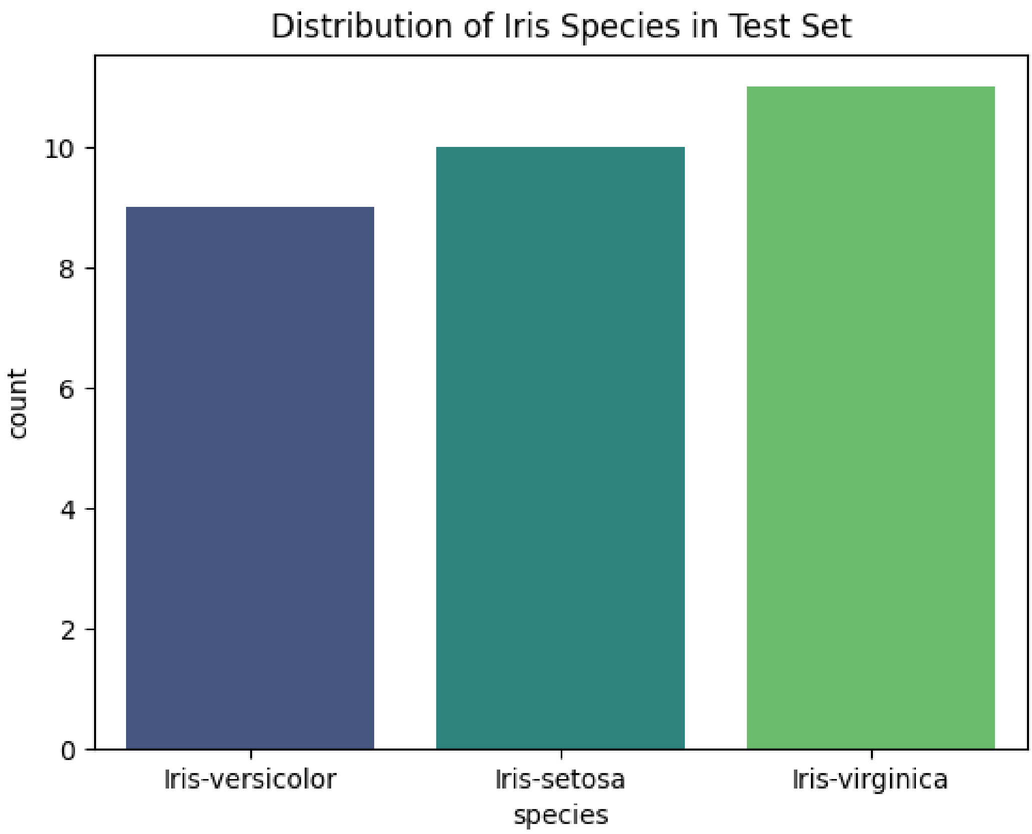
Data split successfully!

Training features shape: (120, 4)

Testing features shape: (30, 4)

Step 4: Visualize Target Distribution in Test Set

```
sns.countplot(x=y_test, hue=y_test, palette="viridis", legend=False)
plt.title("Distribution of Iris Species in Test Set")
plt.show()
```



Step 5: Model Training and Evaluation

Output: Logistic Regression Model:

```
Logistic Regression Model
log_reg_model = LogisticRegression(max_iter=200)  # Initialize Logistic
Regression
log_reg_model.fit(X_train, y_train)  # Train the model
y_pred_lr = log_reg_model.predict(X_test)  # Make predictions on the test set

# Evaluation of Logistic Regression Model
print("\nLogistic Regression Model:")
conf_matrix_lr = confusion_matrix(y_test, y_pred_lr)
print("Confusion Matrix:\n", conf_matrix_lr)
print("Classification Report:\n", classification_report(y_test, y_pred_lr))
accuracy_lr = accuracy_score(y_test, y_pred_lr)
print(f"Logistic Regression Model Accuracy: {accuracy_lr * 100:.2f}%")

Random Forest Model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)  #
Initialize Random Forest
rf_model.fit(X_train, y_train)  # Train the model
y_pred_rf = rf_model.predict(X_test)  # Make predictions on the test set

# Evaluation of Random Forest Model
print("\nRandom Forest Model:")
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
print("Confusion Matrix:\n", conf_matrix_rf)
print("Classification Report:\n", classification_report(y_test, y_pred_rf))
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Random Forest Model Accuracy: {accuracy_rf * 100:.2f}%")
```

Output:

Confusion Matrix:

[[10 0 0]

[0 9 0]

[0 0 11]]

Classification Report:

	precision	**recall**	**f1-score**	**support**
Iris-setosa	1.00	1.00	1.00	10
Iris-versicolor	1.00	1.00	1.00	9
Iris-virginica	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

Logistic Regression Model Accuracy: 100.00%

Random Forest Model:

Confusion Matrix:

[[10 0 0]

[0 9 0]

[0 0 11]]

Classification Report:

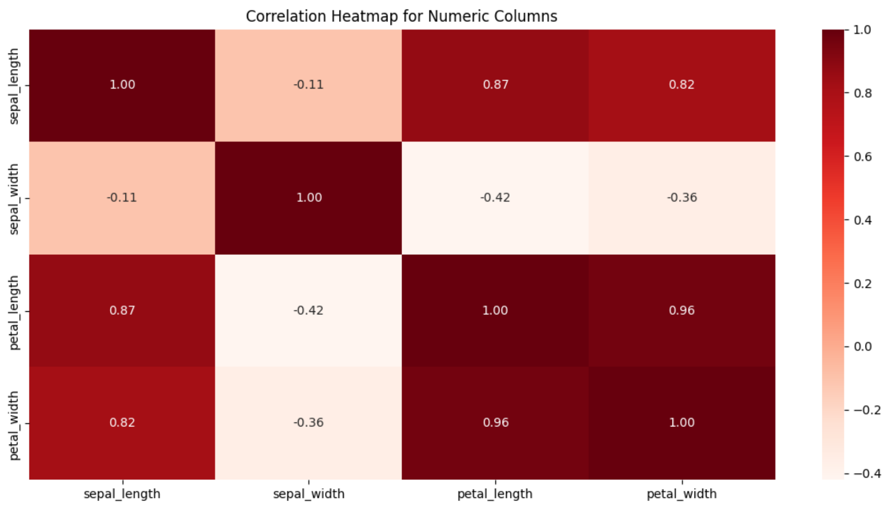
	precision	**recall**	**f1-score**	**support**
Iris-setosa	**1.00**	**1.00**	**1.00**	**10**
Iris-versicolor	**1.00**	**1.00**	**1.00**	**9**
Iris-virginica	**1. 00**	**1.00**	**1.00**	**11**
accuracy			**1.00**	**30**
macro avg	**1.00**	**1.00**	**1.00**	**30**
weighted avg	**1.00**	**1.00**	**1.00**	**30**

Random Forest Model Accuracy: 100.00%

Step 6: Correlation Analysis

```
numeric_df = df.select_dtypes(include=[np.number])  # Select numeric columns
correlation_matrix = numeric_df.corr()  # Compute correlations
```

```
# Plot the correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='Reds', fmt='.2f', cbar=True)
plt.title("Correlation Heatmap for Numeric Columns")
plt.show()
```



Step 7: Prediction on New Data

```
data = {
    'sepal_length': [5],
    'sepal_width': [3],
    'petal_length': [1.5],
    'petal_width': [0.2],
}

input_data = pd.DataFrame(data)  # Convert the dictionary into a DataFrame
flower_type = rf_model.predict(input_data)[0]  # Predict the flower type using
Random Forest
print(f"\nFlower Classified as: {flower_type}")
```

Output:

Flower Classified as: Iris-setosa

Conclusion:

Iris flowers into three species using Logistic Regression and Random Forest models. Random Forest outperformed Logistic Regression with 100% accuracy, showcasing its ability to capture complex patterns in the data. Petal dimensions were found to be key predictors. The study highlights the effectiveness of Random Forest for multi-class classification tasks and suggests further exploration with larger datasets and additional models for enhanced robustness.