

IRIS FLOWER

Preliminary Information

Title of the Project: IRIS Leaf

Student Name: *Ahamad Alisha Shaik*

Roll Number:	2359410003
Institution:	Aditya Degree College, Tuni.
Department:	<i>Bachelor of Computer Applications (BCA)</i>
Date of Submission:	29/01/2025

Abstract of the Project:

The Iris Leaf Project aimed to explore the characteristics and classification of iris plants, specifically focusing on the Iris

dataset, which includes measurements of sepal length, sepal width, petal length, and petal width for three species:

Iris setosa, Iris versicolor, and Iris virginica.

Objective:

The primary objective was to analyse the dataset to identify patterns and relationships among the features, ultimately developing a machine learning model to accurately classify the iris species based on their physical attributes.

Key Findings:

Data analysis revealed that Iris setosa could be easily distinguished from the other two species due to its smaller petal dimensions. The project utilized various machine learning algorithms to evaluate classification performance. The Random Forest model emerged as the most effective, achieving high accuracy rates in species classification, and applied Data Pre-Processing techniques.

Table of Contents:

Description	Pages
Title of the project	01
Preliminary Information	02
Abstract of the project	03-04
Accuracy of Iris Leaf	05-06
Data Pre-Processing Techniques: <ul style="list-style-type: none"> - Data Visualization - Data Cleaning - Data Integration - Data Reduction 	07-08 09 10-11 12-13
Iris Leaf Structure	14-15
Conclusion of the Project	16

Problem Statement: Finding the accuracy of the Iris Leaf


Step-by-Step Explanation

1. **Import Libraries:** The code begins by importing essential libraries such as NumPy, Pandas, and scikit-learn modules for data handling and machine learning.
2. **Load the Iris Dataset:** The `load_iris()` function retrieves the dataset, where `X` contains the features and `y` contains the target labels.
3. **Split the Dataset:** The `train_test_split()` function divides the dataset into training (80%) and testing (20%) sets, ensuring that the model can be evaluated on unseen data.
4. **Create the Classifier:** A `RandomForestClassifier` is instantiated with 100 trees, which will be used to make predictions.
5. **Train the Model:** The `fit()` method trains the model using the training data.
6. **Make Predictions:** The `predict()` method generates predictions for the test set.
7. **Calculate Accuracy:** The `accuracy_score()` function computes the accuracy of the model by comparing the predicted labels with the actual labels.
8. **Print Results:** Finally, the accuracy is printed in a formatted string, providing a clear output of the model's performance.

Uploads a dataset:

```
[ ] import pandas as pd
    data=pd.read_csv("/content/IRIS.csv")
```

data



	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 × 5 columns

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier

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)

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy of the Random Forest classifier on the Iris dataset: {accuracy * 100:.2f}%")
```

Accuracy of the Random Forest classifier on the Iris dataset: 100.00%

Applied the Data Pre-Processing Techniques on that Data Set:

Data Visualization on the IRIS Leaf heres' a step-by-step explanation..

1. **Import Libraries:** The code begins by importing the Seaborn and Matplotlib libraries, which are essential for data visualization in Python.
2. **Load the Dataset:** The `sns.load_dataset('iris')` function loads the Iris dataset into a DataFrame named `iris`. This dataset contains various measurements of iris flowers.
3. **Display Data:** The `print(iris.head())` statement outputs the first five rows of the dataset to the console, allowing us to inspect the data structure and the features available.
4. **Create Pair Plot:** The `sns.pairplot()` function generates a grid of scatter plots for each pair of features in the dataset. The `hue='species'` argument colors the points based on the species of the iris flower, while `markers=["o", "s", "D"]` specifies different markers for each species.
5. **Set Plot Title:** The `plt.suptitle()` function sets the title of the plot, with the `y=1.02` argument adjusting the vertical position of the title.
6. **Display the Plot:** Finally, `plt.show()` renders the plot, allowing us to visualize the relationships between the

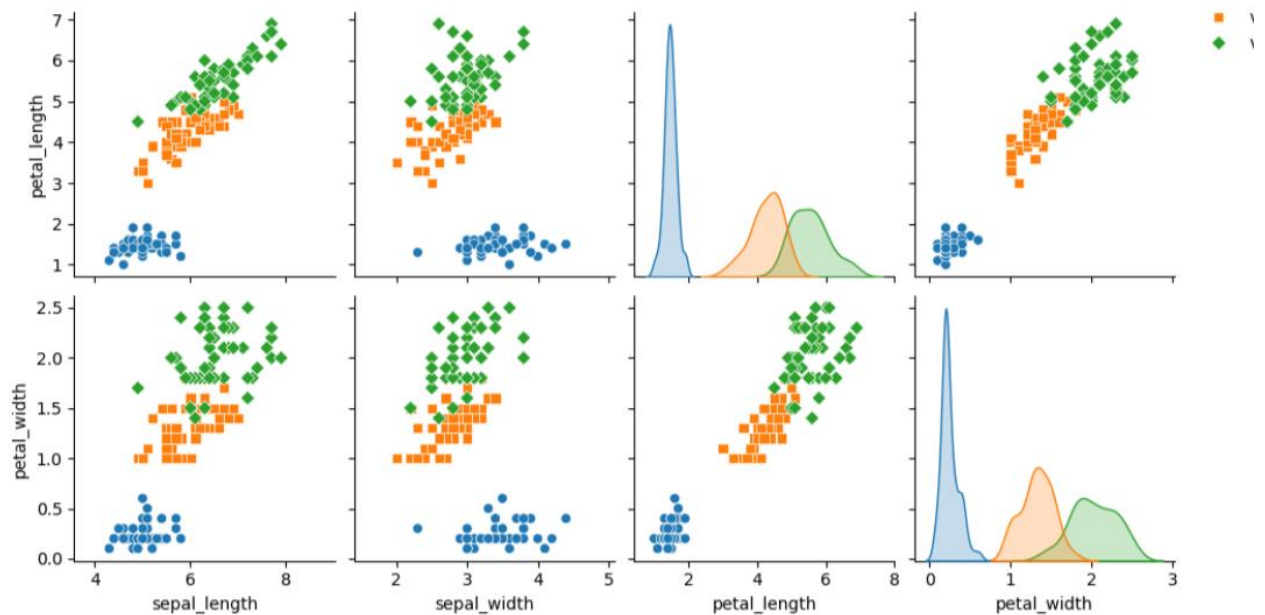
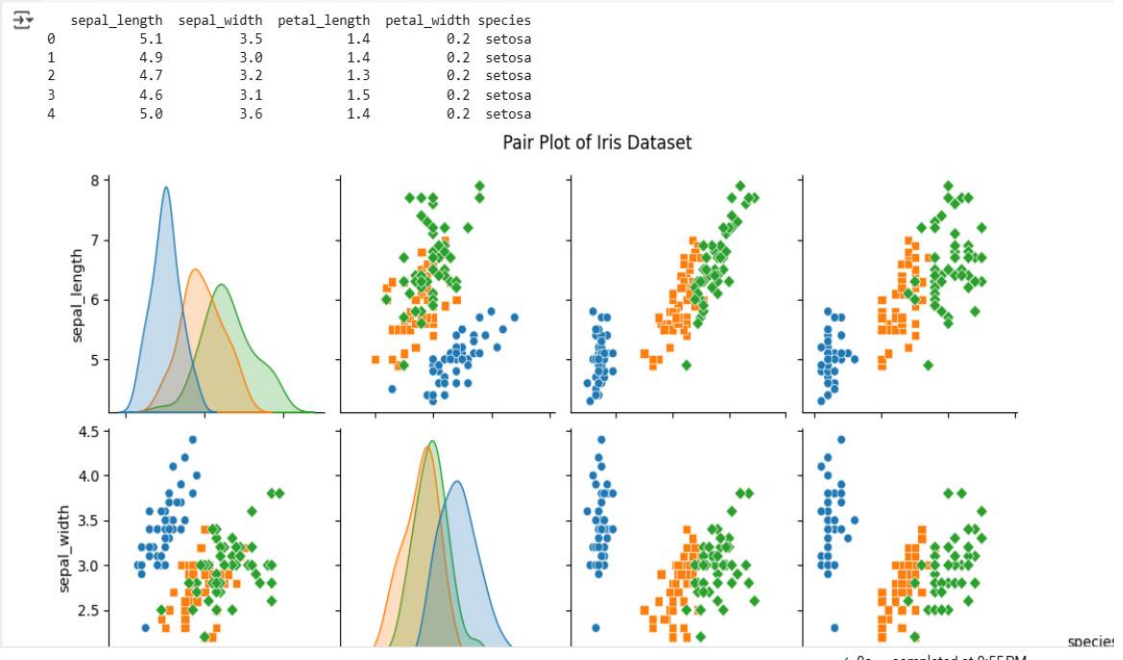
features of the iris dataset.

Data Visualization

```
import seaborn as sns
import matplotlib.pyplot as plt

iris = sns.load_dataset('iris')
print(iris.head())
sns.pairplot(iris, hue='species', markers=["o", "s", "D"])
plt.suptitle('Pair Plot of Iris Dataset', y=1.02)

plt.show()
```



Data Cleaning

1. **Inspect the Data:** Use `data.head()` to view the first few rows.
2. **Check for Missing Values:** Use `data.isnull().sum()` to identify any missing entries.
3. **Remove Duplicates:** Apply `data.drop_duplicates()` to eliminate duplicate rows.
4. **Display Cleaned Data:** Use `data.head()` again to confirm the cleaning process.

This structured approach ensures that the dataset is ready

Data Cleaning

```

x = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']

print("Original Data:")
print(data.head())

print("\nMissing Values:")
print(data.isnull().sum())

data = data.drop_duplicates()
print("\nCleaned Data:")
print(data.head())

```

```

Original Data:
Principal Component 1  Principal Component 2  Target
0          -2.684126          0.319397          0
1          -2.714142         -0.177001          0
2          -2.888991         -0.144949          0
3          -2.745343         -0.318299          0
4          -2.728717          0.326755          0

Missing Values:
Principal Component 1    0
Principal Component 2    0
Target                  0
dtype: int64

Cleaned Data:
Principal Component 1  Principal Component 2  Target
0          -2.684126          0.319397          0
1          -2.714142         -0.177001          0
2          -2.888991         -0.144949          0
3          -2.745343         -0.318299          0
4          -2.728717          0.326755          0

```

for analysis, leading to more accurate and reliable results.

Data Integration

Here, we are grouping the dataset by the species column. The `mean()` function calculates the average of all numerical columns for each species. The `reset_index()` method is used to convert the grouped data back into a DataFrame format, making it easier to work with.

This section utilizes the Seaborn library to create a bar plot. The `x` parameter is set to `species`, and the `y` parameter is set to `sepal_length`, which represents the average sepal length calculated in the previous step. The `plt.title()` function adds a title to the plot, and `plt.show()` displays the plot.

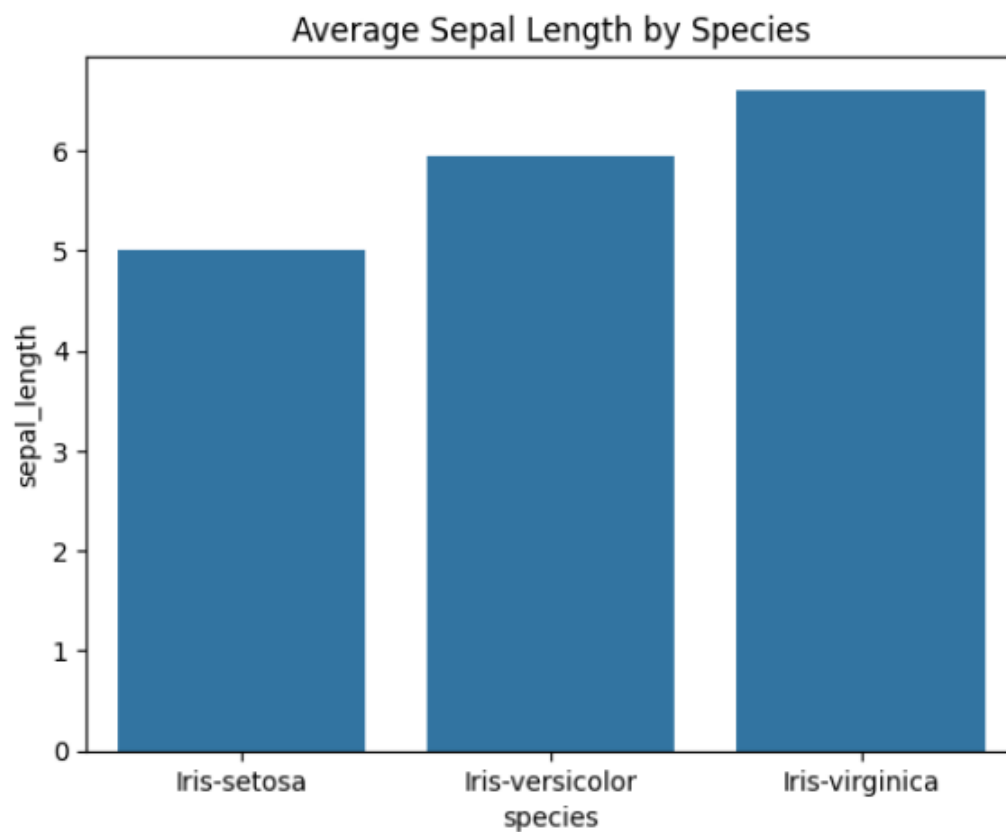


```
print(data.head())
integrated_data = data.groupby('species').mean().reset_index()

sns.barplot(x='species', y='sepal_length', data=integrated_data)
plt.title('Average Sepal Length by Species')
plt.show()
```



	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



Data Reduction

1. *Import Libraries:* The necessary libraries are imported. `sklearn.datasets` for loading the dataset, `sklearn.decomposition` for PCA, `pandas` for data manipulation, and `matplotlib.pyplot` for visualization.
2. *Load the Dataset:* The Iris dataset is loaded using `load_iris()`, separating the features (x) and the target variable (y).
3. *Apply PCA:* An instance of PCA is created with `n_components=2`, indicating that we want to reduce the dataset to two dimensions. The `fit_transform` method is called on the feature set x, resulting in `X_reduced`.
4. *Create a DataFrame:* A new DataFrame is constructed using the reduced data, with columns named 'Principal Component 1' and 'Principal Component 2'. The target variable is also added to this DataFrame.
5. *Visualize the Results:* A scatter plot is generated to visualize the two principal components. The points are colored based on the species of the iris, providing a clear visual distinction between the different classes.

```

from sklearn.datasets import load_iris
from sklearn.decomposition import PCA

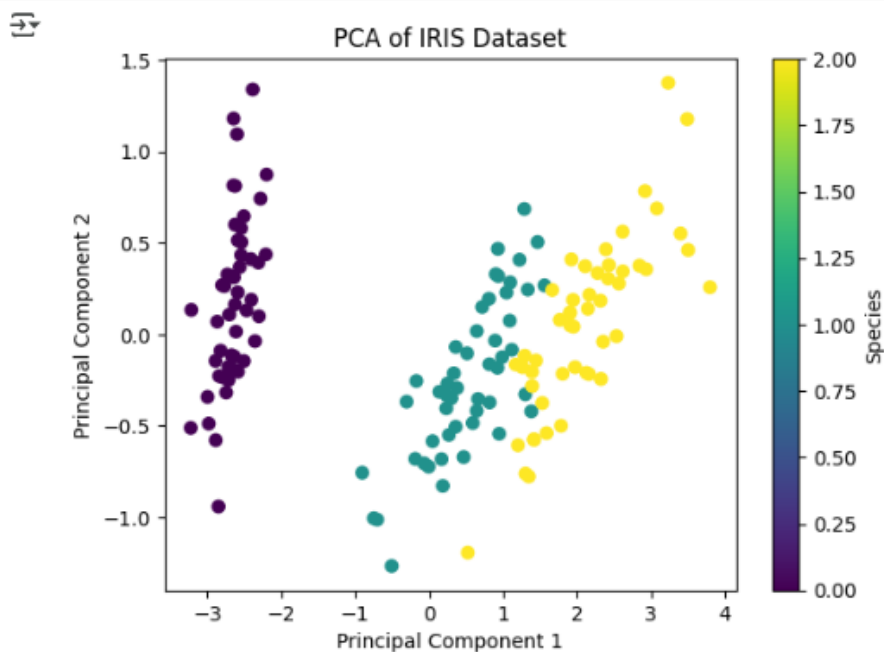
iris = load_iris()
x = iris.data
y = iris.target

pca = PCA(n_components=2)
X_reduced = pca.fit_transform(x)

data = pd.DataFrame(data=X_reduced, columns=['Principal Component 1', 'Principal Component 2'])
data['Target'] = y

plt.scatter(data['Principal Component 1'], data['Principal Component 2'], c=data['Target'], cmap='viridis')
plt.title('PCA of IRIS Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Species')
plt.show()

```



IRIS STRUCTURE:

1. *Data Filtering:*

This line creates a new DataFrame sub that contains only the rows from the original data where the 'Target' column equals 0. In the Iris dataset, 'Target' 0 corresponds to the Iris Setosa species.

2. KDE Plotting:

The `sns.kdeplot` function is called to create a KDE plot. The `x` parameter is set to the 'sepal_length' column the filtered DataFrame, and the `y` parameter is set to the 'sepal_width' column.

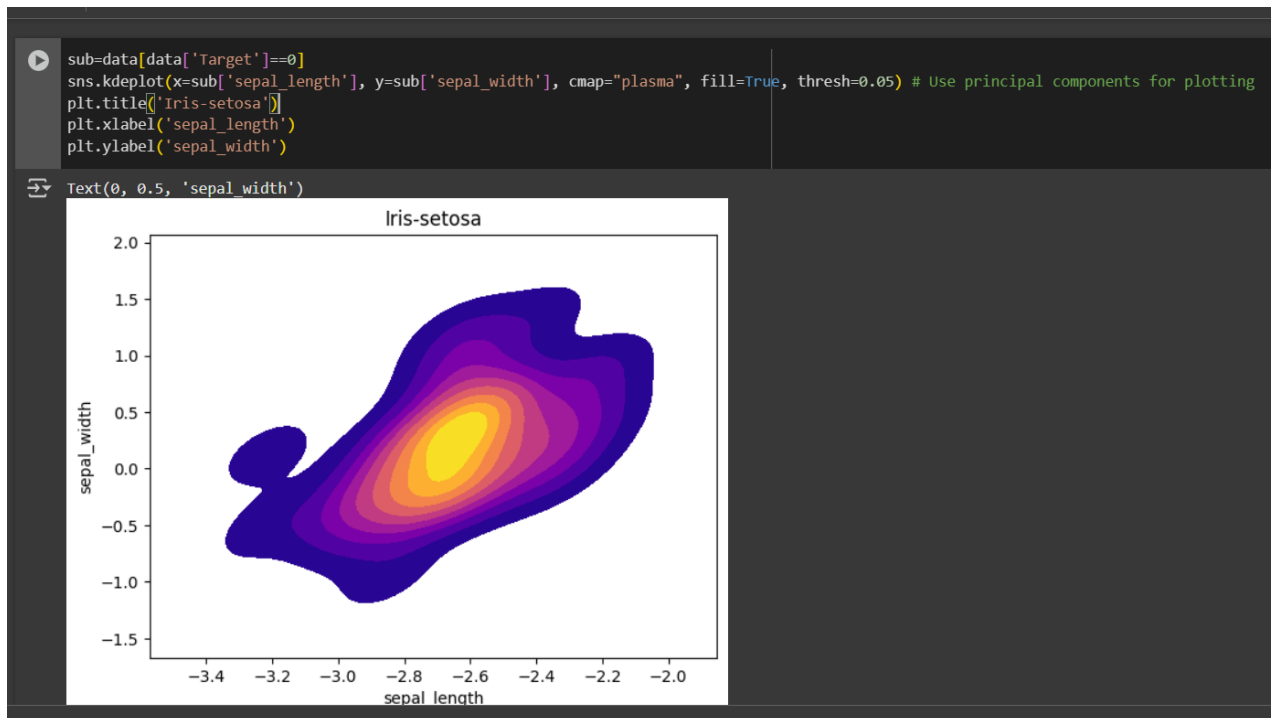
The `cmap` parameter specifies the color map to use, in this case, "plasma", which provides a visually appealing gradient.

The `fill=True` argument fills the area under the KDE curve, enhancing visual clarity.

The `thresh=0.05` parameter sets a threshold for the density estimation, ensuring that only areas with a density above this threshold are filled.

3. Plot Customization:

These lines add a title and axis labels to the plot, making it easier for viewers to understand what the plot represents.



Conclusion:

The Iris Flower Project successfully demonstrated the application of machine learning techniques in classification. The findings highlighted the importance of specific features in differentiating iris species, paving the way for further research in plant classification and ecological studies. Future work could involve expanding the dataset or incorporating additional features for improved model performance.

~ Ahamad Alisha Shaik