Name : Atif ullah khan

Reg : B22F0059AI120

class : Ai Green

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

? Iris Dataset - Target Class Visualization

? Introduction

This document explains the **visualization of the target class (species)** in the **Iris dataset**. We use **Seaborn & Matplotlib** to explore the distribution and variation of species based on different features.

? Understanding the Iris Dataset

The **Iris dataset** contains **150 samples** of **three different species** of flowers:

1. **Setosa**

2. **Versicolor**

3. **Virginica**

Each sample has four numerical features:

- **Sepal Length (cm)**
- **Sepal Width (cm)**
- **Petal Length (cm)**
- **Petal Width (cm)**

The dataset is widely used for **classification tasks** in machine learning.

? Target Class Visualization

? 1. Countplot of Species Distribution

- This plot **counts the number of samples** for each species.
- It helps us understand if the dataset is **balanced**.
- Since the dataset is well-structured, each species has an **equal number of samples (50 each).**

? 2. Boxplot of Petal Length by Species

- A **boxplot** shows the **distribution of petal length** across different species.
- The **box** represents the **interquartile range (IQR)**, showing how petal length varies within each species.
- **Setosa** has the **smallest petal length**, while **Virginica** has the **largest.**
- This helps in identifying **outliers** and understanding **feature importance**.

? 3. Violin Plot of Sepal Width by Species

- A **violin plot** combines a boxplot with a **density estimation**.
- It shows the **distribution shape** of **sepal width** for each species.
- Setosa has a **higher density** in certain sepal width ranges.
- This plot helps in understanding the **spread and distribution of features**.

? Key Insights from the Visualizations

✓ The dataset is **balanced**, with 50 samples per species.
✓ **Petal length** is a **strong distinguishing feature** among species.
✓ **Sepal width distribution** varies significantly for different species.
✓ **Setosa is easily separable**, while Versicolor and Virginica show some overlap.

These visualizations provide **valuable insights** for classification models, helping in **feature selection and pattern recognition**.

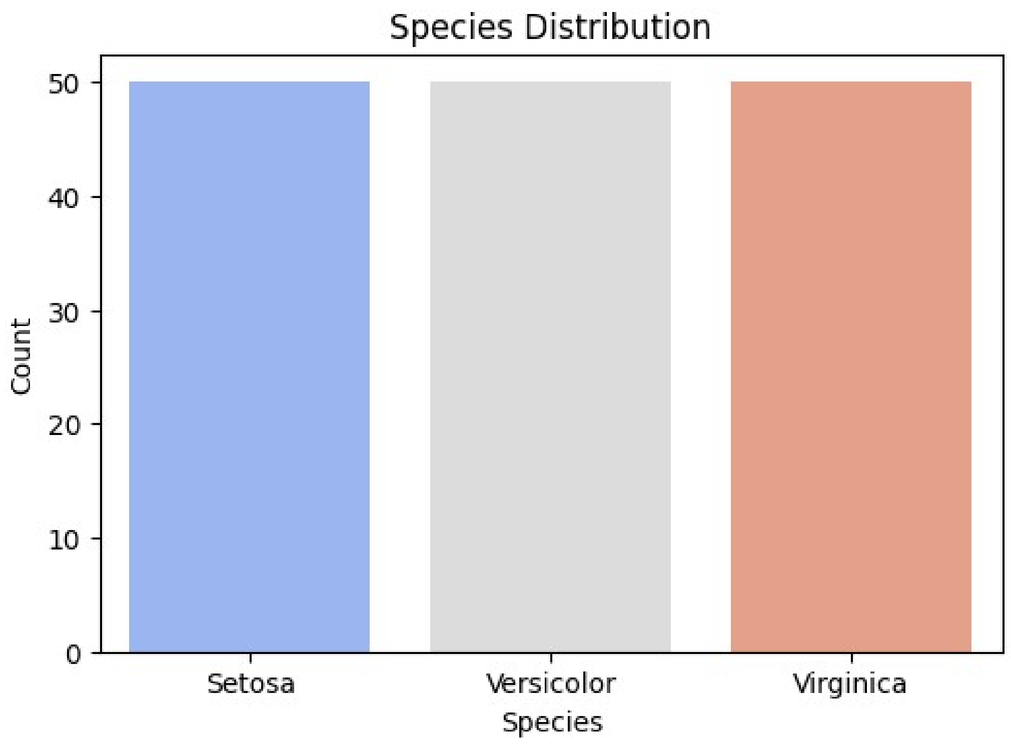
▯ Conclusion

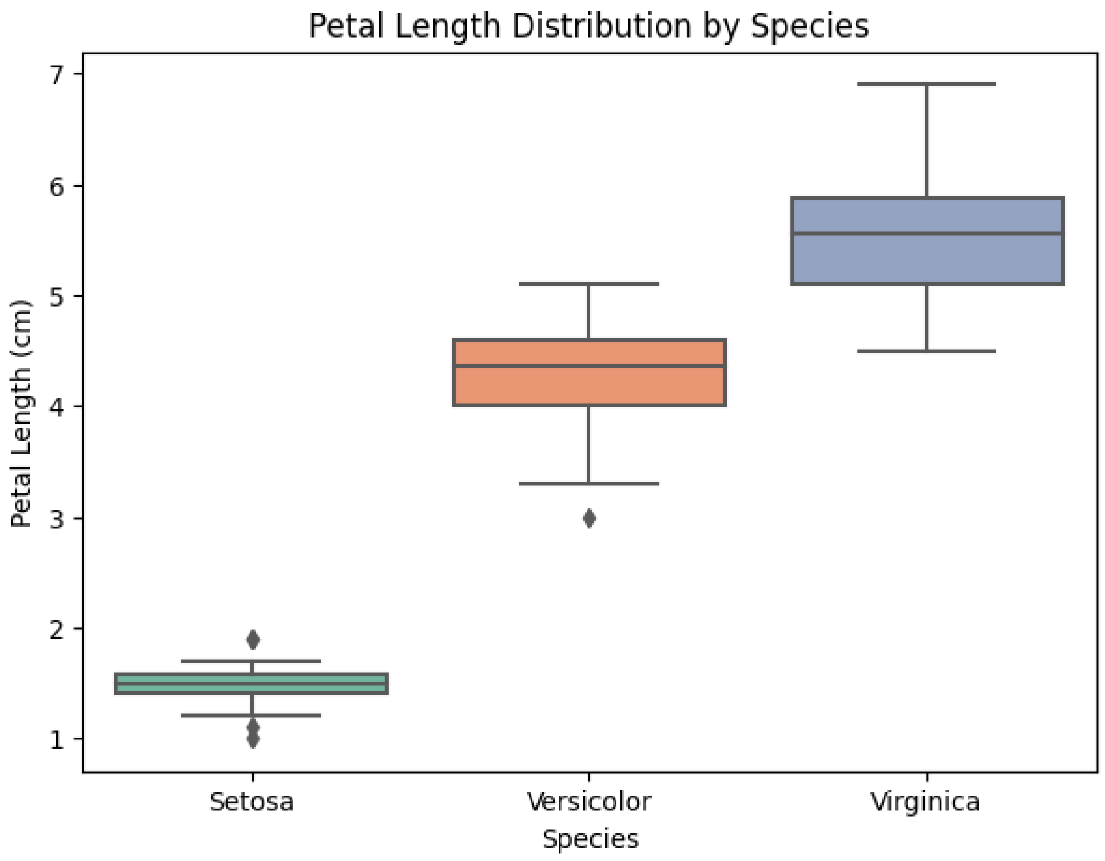
Visualizing the **target class (species)** in the **Iris dataset** is crucial for understanding data patterns.
These plots reveal important relationships that can be leveraged in **machine learning models**.

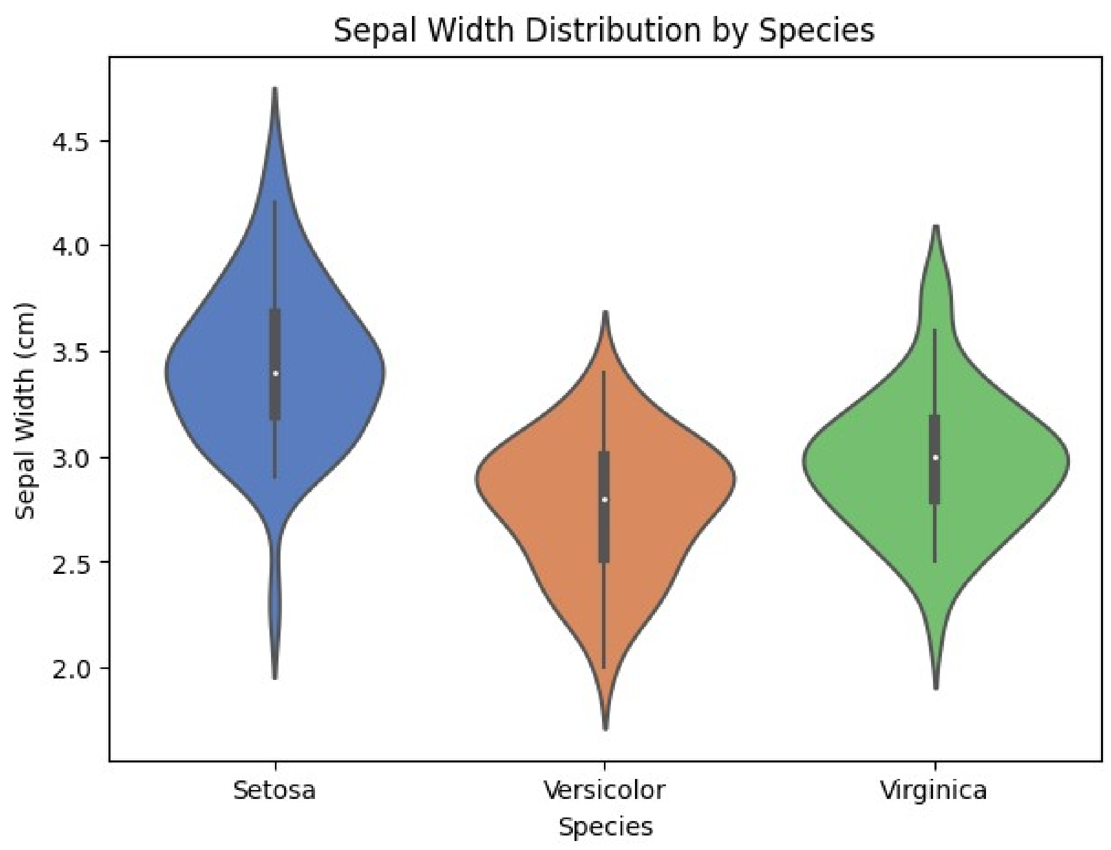
By analyzing **species distribution, petal length variations, and sepal width density**, we gain a deeper understanding of how different features distinguish **Iris flower species**.

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, ClassifierMixin
import matplotlib.pyplot as plt
import numpy as np

# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target

# Map target values to species names
df['species'] = df['species'].map({0: 'Setosa', 1: 'Versicolor', 2:
'Virginica'})

# ▯ Plot 1: Countplot of species distribution
plt.figure(figsize=(6, 4))
sns.countplot(x=df['species'], palette='coolwarm')
plt.title("Species Distribution")
plt.xlabel("Species")
plt.ylabel("Count")
plt.show()
```

```
# 📦 Plot 2: Boxplot of petal length by species
plt.figure(figsize=(7, 5))
sns.boxplot(x='species', y='petal length (cm)', data=df,
palette='Set2')
plt.title("Petal Length Distribution by Species")
plt.xlabel("Species")
plt.ylabel("Petal Length (cm)")
plt.show()

# 🎻 Plot 3: Violin plot of sepal width by species
plt.figure(figsize=(7, 5))
sns.violinplot(x='species', y='sepal width (cm)', data=df,
palette='muted')
plt.title("Sepal Width Distribution by Species")
plt.xlabel("Species")
plt.ylabel("Sepal Width (cm)")
plt.show()
```




Petal Length Distribution by Species
Petal Length (cm)
1
2
3
4
5
6
7
Setosa
Versicolor
Virginica
Species



```python
# Load the dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split dataset into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

class ANN(nn.Module):
    def __init__(self):
        super(ANN, self).__init__()
        self.fc1 = nn.Linear(4, 16)  # 4 input features -> 16 neurons
        self.bn1 = nn.BatchNorm1d(16)
        self.dropout1 = nn.Dropout(0.3)

        self.fc2 = nn.Linear(16, 8)  # 16 -> 8 neurons
        self.bn2 = nn.BatchNorm1d(8)
        self.dropout2 = nn.Dropout(0.3)

        self.fc3 = nn.Linear(8, 3)  # 8 -> 3 output classes

        self.activation = nn.ReLU()

        # Initialize weights
        nn.init.kaiming_normal_(self.fc1.weight, nonlinearity='relu')
        nn.init.kaiming_normal_(self.fc2.weight, nonlinearity='relu')
        nn.init.xavier_normal_(self.fc3.weight)

    def forward(self, x):
        x = self.activation(self.bn1(self.fc1(x)))
        x = self.dropout1(x)
        x = self.activation(self.bn2(self.fc2(x)))
        x = self.dropout2(x)
        x = self.fc3(x)  # No activation (CrossEntropyLoss applies
Softmax internally)
        return x

class PyTorchClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, lr=0.01, epochs=100, batch_size=16):
        self.lr = lr
        self.epochs = epochs
        self.batch_size = batch_size
        self.model = ANN()
        self.criterion = nn.CrossEntropyLoss()
        self.optimizer = optim.Adam(self.model.parameters(),
lr=self.lr)
        self.train_losses = []

    def fit(self, X, y):
        X_tensor = torch.FloatTensor(X)
        y_tensor = torch.LongTensor(y)

        for epoch in range(self.epochs):
            self.optimizer.zero_grad()
            outputs = self.model(X_tensor)
            loss = self.criterion(outputs, y_tensor)
            loss.backward()
            self.optimizer.step()
            self.train_losses.append(loss.item())

            if (epoch + 1) % 10 == 0:
                print(f"Epoch {epoch+1}/{self.epochs}, Loss:
{loss.item():.4f}")

        return self

    def predict(self, X):
        X_tensor = torch.FloatTensor(X)
        outputs = self.model(X_tensor)
        _, predictions = torch.max(outputs, 1)
        return predictions.numpy()

    def score(self, X, y):
        predictions = self.predict(X)
        return np.mean(predictions == y)
```

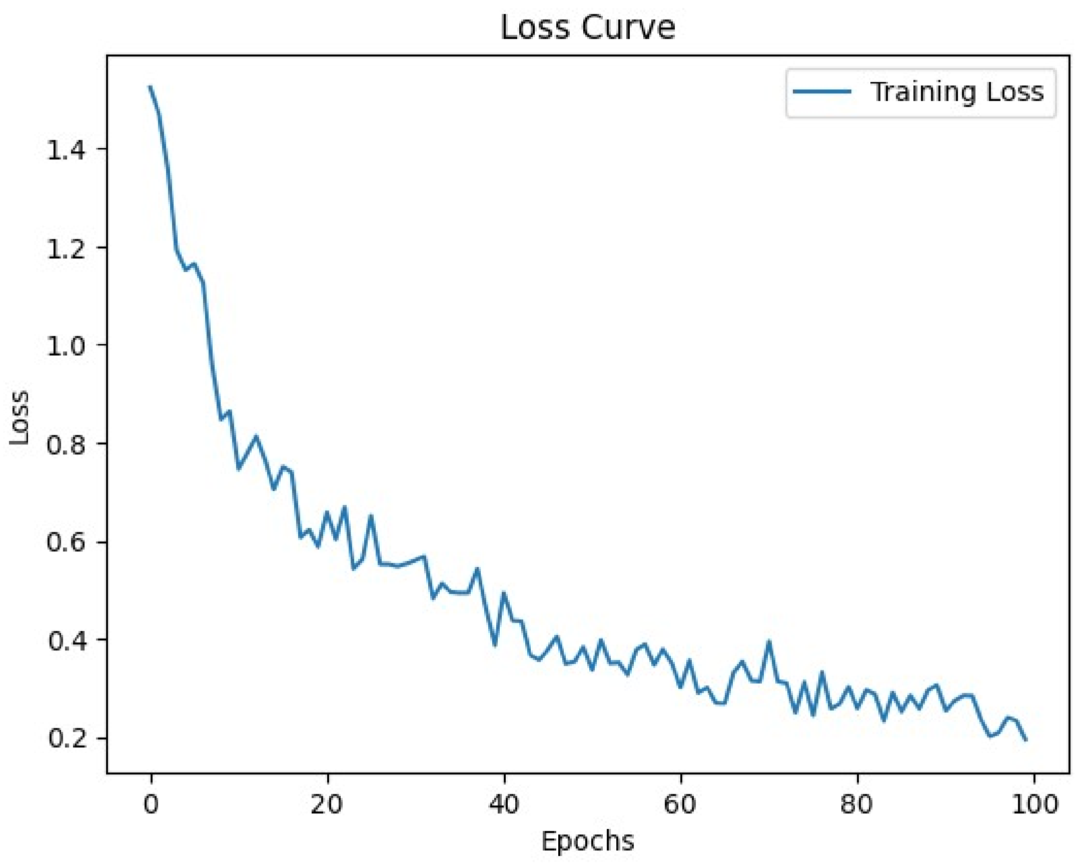
```
pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Feature scaling
    ('classifier', PyTorchClassifier(lr=0.01, epochs=100))  # Train
ANN
])
```

```
# Train the pipeline
pipeline.fit(X_train, y_train)
```

```
# Evaluate the model
accuracy = pipeline.score(X_test, y_test)
print(f"Test Accuracy: {accuracy:.2f}")
```

```
Epoch 10/100, Loss: 0.8638
Epoch 20/100, Loss: 0.5876
Epoch 30/100, Loss: 0.5534
Epoch 40/100, Loss: 0.3870
Epoch 50/100, Loss: 0.3837
Epoch 60/100, Loss: 0.3505
Epoch 70/100, Loss: 0.3138
Epoch 80/100, Loss: 0.3028
Epoch 90/100, Loss: 0.3065
Epoch 100/100, Loss: 0.1953
Test Accuracy: 0.83
```

```
plt.plot(pipeline.named_steps['classifier'].train_losses,
label="Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss Curve")
plt.legend()
plt.show()
```


Loss Curve
Training Loss
Loss
Epochs
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100