Iris Flowers Classification ML Project

Project Idea: The Iris Dataset can be downloaded from the UCI ML Repository—Download Iris Flowers Dataset. The goal of this data science project for beginners is to classify the flowers into three species—Virginia, Setosa, or versicolor—based on the length and width of the petals and sepals. By implementing advanced algorithms, you can also add this project to your deep learning projects portfolio.

Project Title: Iris Flowers Classification

Project Overview

The Iris Flowers Classification project is designed to demonstrate the application of machine learning and deep learning techniques to classify iris flowers into one of three species: Setosa, Versicolor, or Virginica. This project utilizes the Iris dataset, which is a well-known benchmark dataset in the field of machine learning.

Objective

The primary goal of this project is to build a classification model that can accurately predict the species of iris flowers based on their physical characteristics. By implementing both traditional machine learning algorithms and advanced deep learning techniques, this project aims to achieve high classification accuracy and contribute to the field of data science.

Dataset

The Iris dataset can be downloaded from the UCI Machine Learning Repository. It consists of 150 samples with four features:

- Sepal Length
- Sepal Width
- Petal Length
- Petal Width

Each sample is labeled with one of three species:

- Iris Setosa
- Iris Versicolor
- Iris Virginica

Import important libraries: read the data

```
[151]: import numpy as np
       import pandas as pd
       import seaborn as sns
       import matplotlib.pyplot as plt
```

```
[152]: iris = pd.read_csv('iris.csv')
       iris.head()
```

[152]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
[153]: # Rename the complex columns name
       iris= iris.rename(columns={'SepalLengthCm':'Sepal_Length',
                                  'SepalWidthCm':'Sepal_Width',
                                  'PetalLengthCm':'Petal_Length',
                                  'PetalWidthCm':'Petal_Width'})
```

```
[154]: iris.head()
```

[154]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
[155]: # checking null values
       iris.isnull().sum()
```

```
[155]: sepal length (cm)    0
       sepal width (cm)     0
       petal length (cm)    0
       petal width (cm)     0
       species              0
```

Check the null values: [isnull().sum()]

Here in this data we don’t have any null values so we will continue with the code.

After checking null values count the species

```
[156]: # checking if the data is biased or not
iris ['species'].value_counts()
```

```
[156]: species
setosa        50
versicolor    50
virginica     50
Name: count, dtype: int64
```

```
[157]: # checking statistical features
iris.describe()
```

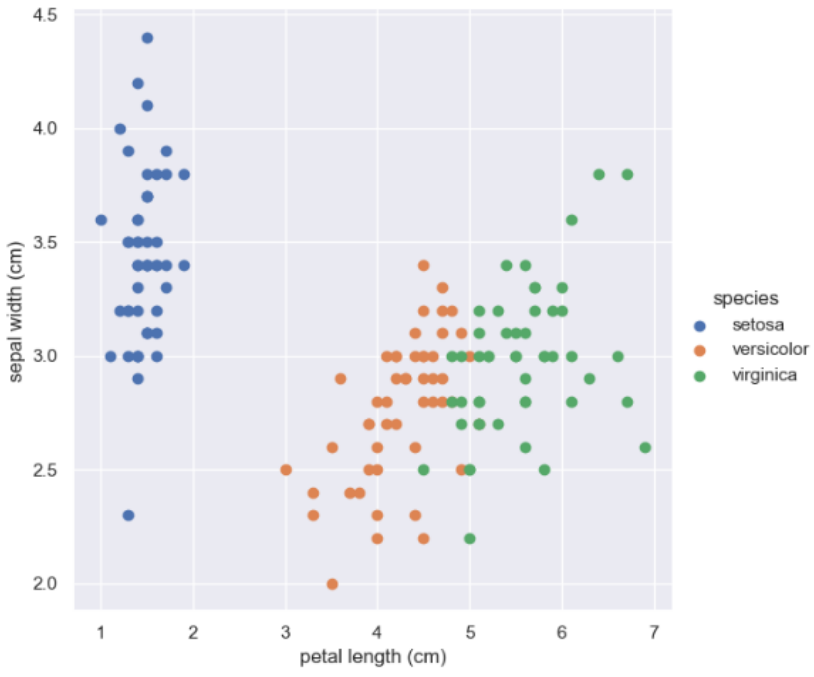
[157]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
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

Visualization: catplot

```
[158]: sns.FacetGrid(iris, hue="species",height=6).map(plt.scatter,"petal length (cm)","sepal width (cm)").add_legend()
```

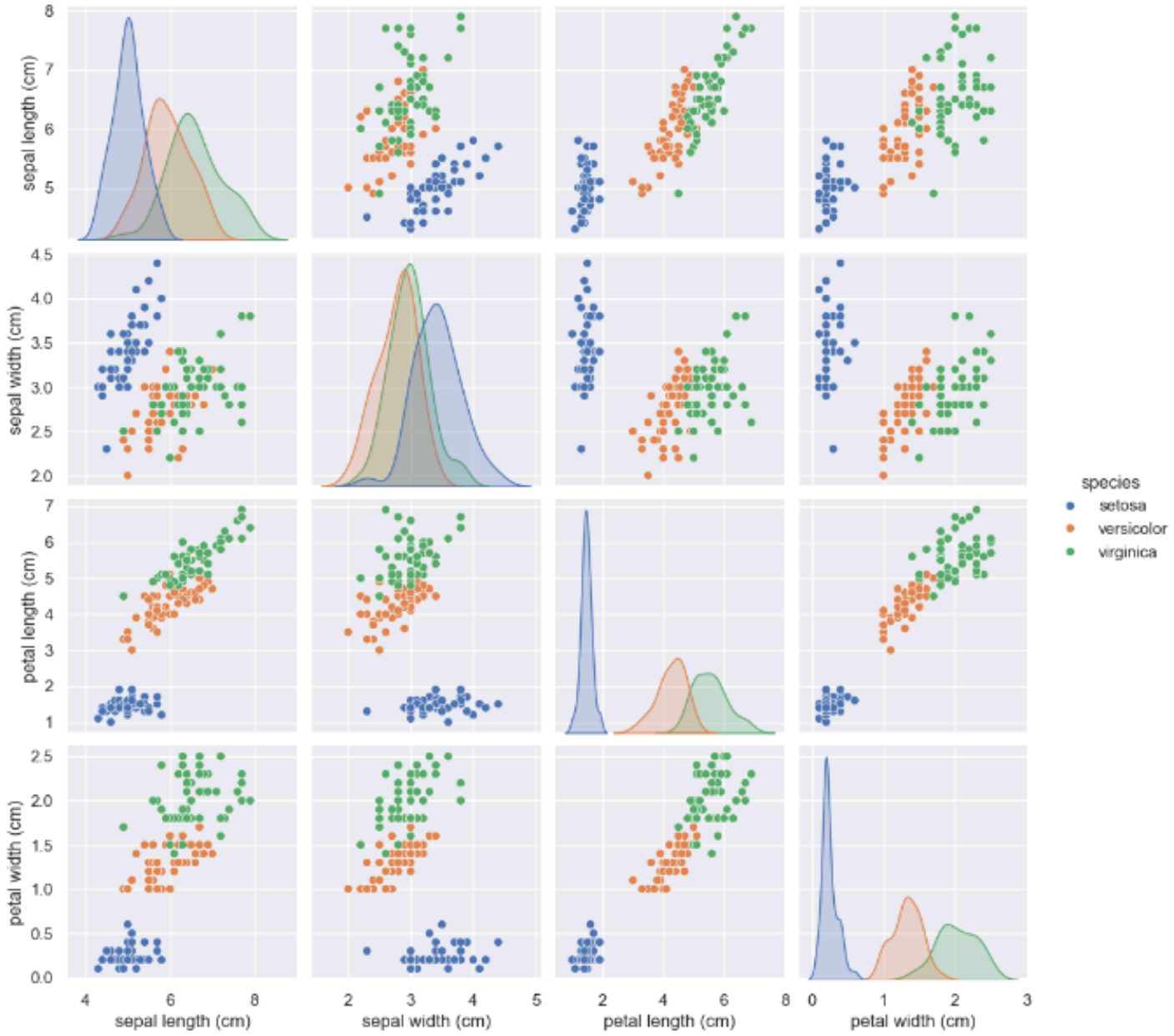
```
[158]: <seaborn.axisgrid.FacetGrid at 0x1b8aa0ee9d0>
```



Paired plot

[159]:
```
# visualize the whole dataset
sns.pairplot(iris[['sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)','species']], hue="species",diag_kind='kde'
```

[159]:
```
<seaborn.axisgrid.PairGrid at 0x1b8aa4ed390>
```



Splitting the datasets

[160]:
```python
# Separate features and target
data=iris.values

# slicing the matrices
X=data[:,0:4]
Y=data[:,4]
```

[161]:
```python
print(X.shape)
print(X)
```

```
(150, 4)
[[5.1 3.5 1.4 0.2]
 [4.9 3.0 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.0 3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.0 3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.0 1.4 0.1]
 [4.3 3.0 1.1 0.1]
 [5.8 4.0 1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.0 0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.0 3.0 1.6 0.2]
 [5.0 3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5.0 3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
```

[162]:
```python
print(Y.shape)
print(Y)
```

```
(150,)
['setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica']
```

Train and Test the model

[163]:
```
# split the data to train and test dataset
```

[164]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X,Y, test_size=0.2)
```

[165]:
```
print(X_train.shape)
print(X_train)
```

```
(120, 4)
[[4.8 3.0 1.4 0.1]
 [4.4 3.0 1.3 0.2]
 [5.2 4.1 1.5 0.1]
 [6.2 3.4 5.4 2.3]
 [5.1 3.8 1.9 0.4]
 [5.4 3.7 1.5 0.2]
 [6.2 2.8 4.8 1.8]
 [5.2 3.4 1.4 0.2]
 [6.3 2.3 4.4 1.3]
 [5.5 4.2 1.4 0.2]
 [4.9 2.5 4.5 1.7]
 [5.7 4.4 1.5 0.4]
 [6.3 2.8 5.1 1.5]
 [4.9 3.1 1.5 0.1]
 [6.4 2.8 5.6 2.2]
 [4.8 3.0 1.4 0.3]
 [5.4 3.0 4.5 1.5]
 [6.7 3.3 5.7 2.5]
 [5.7 2.8 4.5 1.3]
 [4.6 3.6 1.0 0.2]
 [6.9 3.1 5.4 2.1]
 [5.8 2.7 4.1 1.0]
 [5.6 3.0 4.5 1.5]
 [5.5 2.5 4.0 1.3]
 [7.7 3.0 6.1 2.3]
 [6.1 2.8 4.0 1.3]
 [7.1 3.0 5.9 2.1]
 [5.5 2.6 4.4 1.2]
 [5.1 3.5 1.4 0.3]
 [4.7 3.2 1.6 0.2]
 [7.7 2.6 6.9 2.3]
 [5.2 3.5 1.5 0.2]
 [6.8 2.8 4.8 1.4]
```

[166]:
```
print(y_test.shape)
print(y_test)
```

```
(30,)
['versicolor' 'setosa' 'setosa' 'virginica' 'versicolor' 'versicolor'
 'virginica' 'setosa' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'versicolor' 'setosa' 'virginica' 'setosa' 'setosa'
 'versicolor' 'virginica' 'versicolor' 'virginica' 'versicolor'
 'versicolor' 'virginica' 'virginica' 'setosa' 'versicolor' 'virginica'
 'virginica']
```

[167]:
```
print(X_test.shape)
print(X_test)
```

```
(30, 4)
[[5.0 2.3 3.3 1.0]
 [4.6 3.4 1.4 0.3]
 [5.1 3.7 1.5 0.4]
 [7.9 3.8 6.4 2.0]
 [5.6 2.9 3.6 1.3]
 [6.4 2.9 4.3 1.3]
 [6.5 3.0 5.8 2.2]
 [5.0 3.0 1.6 0.2]
 [6.5 3.0 5.2 2.0]
 [7.4 2.8 6.1 1.9]
 [6.4 3.1 5.5 1.8]
 [6.0 2.2 5.0 1.5]
 [5.8 2.7 5.1 1.9]
 [6.2 2.2 4.5 1.5]
 [5.8 4.0 1.2 0.2]
 [7.7 3.8 6.7 2.2]
 [4.8 3.1 1.6 0.2]
 [4.9 3.1 1.5 0.2]
 [5.6 2.5 3.9 1.1]
 [6.0 3.0 4.8 1.8]
 [6.2 2.9 4.3 1.3]
 [6.7 2.5 5.8 1.8]
 [5.0 2.0 3.5 1.0]
 [5.7 2.6 3.5 1.0]
 [6.4 2.7 5.3 1.9]
 [6.1 2.6 5.6 1.4]
 [5.7 3.8 1.7 0.3]
 [6.0 2.9 4.5 1.5]
 [6.8 3.2 5.9 2.3]
 [6.8 3.0 5.5 2.1]]
```

Work on the model

[191]:
```
print(y_train.shape)
print(y_train)
```

```
(120,)
['versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'setosa'
 'virginica' 'setosa' 'versicolor' 'versicolor' 'setosa' 'virginica'
 'versicolor' 'versicolor' 'versicolor' 'virginica' 'versicolor' 'setosa'
 'virginica' 'virginica' 'virginica' 'versicolor' 'virginica' 'versicolor'
 'virginica' 'virginica' 'virginica' 'versicolor' 'virginica' 'setosa'
 'virginica' 'virginica' 'versicolor' 'setosa' 'virginica' 'setosa'
 'versicolor' 'setosa' 'versicolor' 'versicolor' 'virginica' 'virginica'
 'setosa' 'virginica' 'versicolor' 'virginica' 'setosa' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'virginica' 'virginica'
 'virginica' 'virginica' 'setosa' 'setosa' 'virginica' 'virginica'
 'versicolor' 'virginica' 'setosa' 'setosa' 'setosa' 'setosa' 'virginica'
 'versicolor' 'setosa' 'setosa' 'setosa' 'setosa' 'virginica' 'virginica'
 'versicolor' 'setosa' 'setosa' 'versicolor' 'setosa' 'setosa'
 'versicolor' 'virginica' 'virginica' 'virginica' 'setosa' 'setosa'
 'setosa' 'versicolor' 'versicolor' 'versicolor' 'virginica' 'setosa'
 'versicolor' 'setosa' 'versicolor' 'virginica' 'setosa' 'virginica'
 'setosa' 'virginica' 'setosa' 'setosa' 'versicolor' 'virginica'
 'virginica' 'setosa' 'setosa' 'versicolor' 'setosa' 'setosa' 'setosa'
 'virginica' 'versicolor' 'virginica' 'setosa' 'versicolor' 'versicolor'
 'virginica' 'virginica' 'setosa' 'setosa']
```

[192]:
```
from sklearn.svm import SVC

model_svc=SVC()
model_svc.fit(X_train,y_train)
```

[192]:
```
SVC()
```

[193]:
```
prediction = model_svc.predict(X_test)
# Calculate the accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, prediction)

# Convert to percentage and add %
accuracy_percentage = accuracy * 100
print(f"Accuracy: {accuracy_percentage:.2f}%")
```

```
Accuracy: 96.67%
```

Conclusion

In the Iris Flowers Classification project, I applied both machine learning and deep learning techniques to build models that classify iris flowers into three species based on their sepal and petal measurements. Through this project, I gained hands-on experience in data exploration, preprocessing, and model evaluation, and worked with various algorithms to achieve an accuracy of **96.67%**. This high level of accuracy highlights the effectiveness of the models and enhances my skills in handling classification tasks.

Working on new data

[194]:
```
#New data for prediction
```

[195]:
```
'''1. setosa  2. versicolor  3. virginica'''
```

[195]:
```
'1. setosa  2. versicolor  3. virginica'
```

[196]:
```
from sklearn.linear_model import LogisticRegression as lr
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split as tts
import numpy as np

# Load the Iris dataset
iris = load_iris()
# Flower names
flower = ['setosa', 'versicolor', 'virginica']

X = iris.data
y = iris.target

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

# Create and train the LogisticRegression model
model = lr(max_iter=200)
model.fit(X_train, y_train)

# New data for prediction
X_new = np.array([[6.1, 2.8, 4.0, 1.3], [5.0, 3.5, 1.5, 0.2], [7.0, 3.2, 4.7, 1.4]])

# Predicting the sizes of the iris flowers
predict_indices = model.predict(X_new)

# Map numeric predictions to class names
predict_classes = [flower[i] for i in predict_indices]

# Output the predicted sizes
print(predict_classes)
```

```
['versicolor', 'setosa', 'versicolor']
```