K-Nearest-Neighbors Iris Flower Classification Project

Nawal Obaid

9/6/2022

This project uses a built-in seaborn dataset called Iris. It contains 3 classes of 50 instances each, where each class refers to a type of iris plant. The attribute to be predicted is the class of iris plant. The classes are as follows: 1. Iris Setosa, 2. Iris Versicolour, 3. Iris Virginica

There are 4 features:

1. sepalLength: sepal length in cm
2. sepalWidth: sepal width in cm
3. petalLength: petal length in cm
4. petalWidth: petal width in cm

There are 3 classes represneting class label of iris flower {1,2,3}

1. Iris Setosa
2. Iris Versicolour
3. Iris Virginica

image

In [233…

```
# import image module
from IPython.display import Image
# get the image
Image(url="iris_flower.png", width=800, height=800)
```

Out[233]:

Import the needed libraries

In [234…

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

In [235…

```
iris = sns.load_dataset('iris')
```

In [236…

```
iris.head()
# No need for data normalization
```

Out[236]:

	sepal_length	sepal_width	petal_length	petal_width	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

In [237…

```
iris.info()
```

```
<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
```

In [238…

```
iris.describe()
```

Out[238]:

	sepal_length	sepal_width	petal_length	petal_width
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

In [239…

```
iris['species'].unique()
```

Out[239]:

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

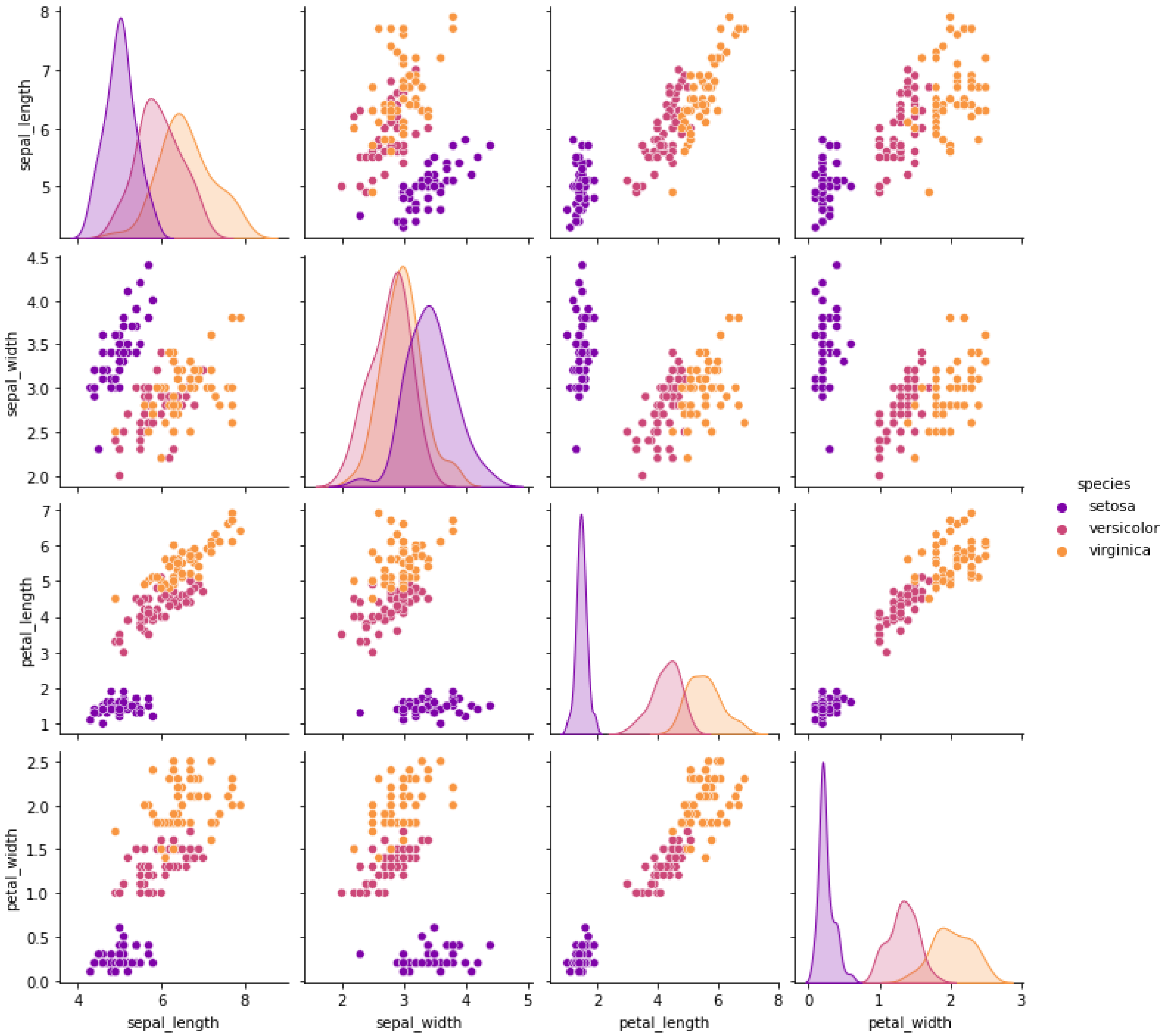
Exploratory Data Analysis

In [240…

```
# Which flower species seems to be the most separable?
# Answer is: A) Setosa
sns.pairplot(data=iris, hue='species', palette = 'plasma')
```

Out[240]:

```
<seaborn.axisgrid.PairGrid at 0x18b955f4ca0>
```



In [241…

```
corr = iris.corr()
```

In [242…

```
sns.heatmap(corr, annot = True, cmap = 'Purples')
```

Out[242]:

```
<AxesSubplot:>
```

In [243…

```
plt.figure(figsize =(15,25))

plt.subplot(4,4,1)
```

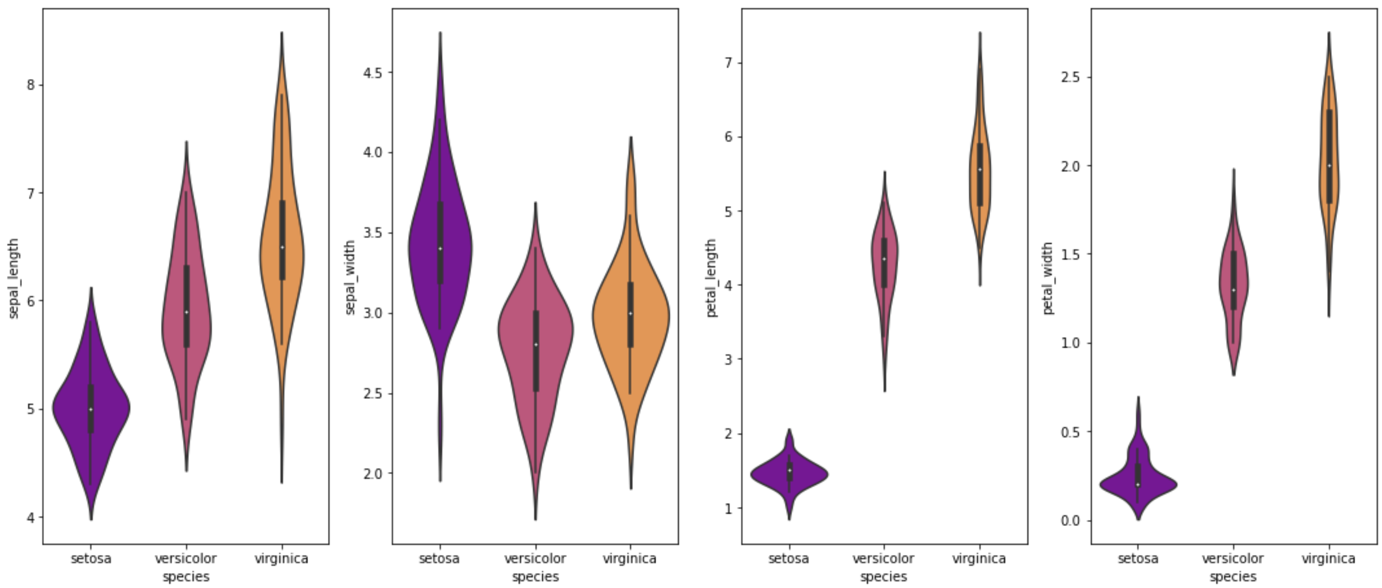
```
sns.violinplot(y = iris['sepal_length'], x= iris['species'], palette = 'plasma')

plt.subplot(4,4,2)
sns.violinplot(y = iris['sepal_width'], x= iris['species'], palette = 'plasma')

plt.subplot(4,4,3)
sns.violinplot(y = iris['petal_length'], x= iris['species'], palette = 'plasma')

plt.subplot(4,4,4)
sns.violinplot(y = iris['petal_width'], x= iris['species'], palette = 'plasma')

plt.tight_layout()
```

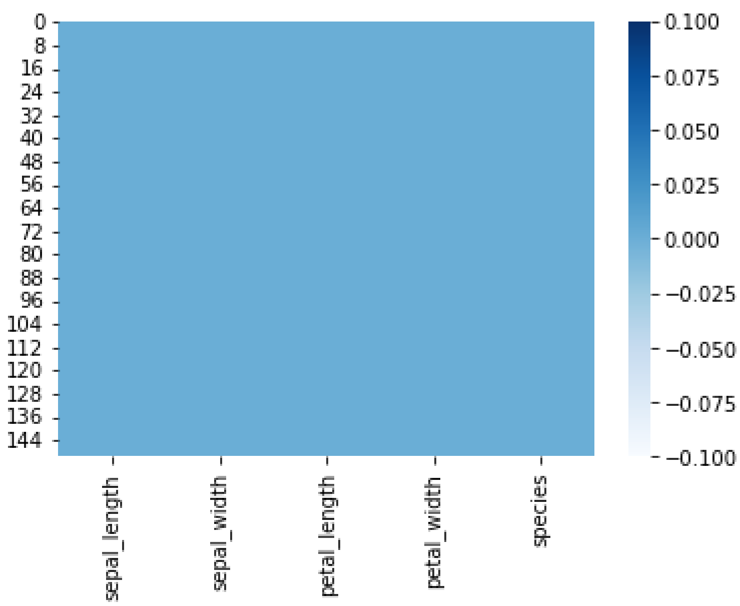


In [244…:

```
sns.heatmap(iris.isnull(), fmt="d", cmap='Blues')
# No missing values
```

Out[244]:

```
<AxesSubplot:>
```



Train Test Split

In [245…:

```
X = iris.drop('species', axis = 1)
y = iris['species']
```

In [246…:

```
from sklearn.preprocessing import LabelEncoder
labelencoder_y = LabelEncoder()
```

```
y = labelencoder_y.fit_transform(y)
```

In [247…
```
y
```

Out[247]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [248…
```
from sklearn.model_selection import train_test_split
```

In [249…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [250…
```
#p : int, default=2 , Power parameter for the Minkowski metric.
#metric : str or callable, default='minkowski'. The default metric is minkowski
#, and with p=2 is equivalent to the standard Euclidean metric.
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5, metric = 'minkowski', p=2)
```

In [251…
```
classifier = KNeighborsClassifier()
```

In [252…
```
# train/fit the classifier to the dataset
classifier.fit(X_train, y_train)
```

Out[252]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [253…
```
# test the classifer and see the predictions
predictions = classifier.predict(X_test)
```

Model Evaluation

In [254…
```
# let's see the performance of this classifier using confusion atrix and classification
from sklearn.metrics import classification_report, confusion_matrix
```

In [255…
```
print('**************Classification Report**************************')
print(classification_report(y_test, predictions))
print('***************Confusion Matrix***************************')
print(confusion_matrix(y_test, predictions))
```

```
**************Classification Report**************************
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      0.92      0.96        12
           2       0.90      1.00      0.95         9

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30

***************Confusion Matrix***************************
[[ 9  0  0]
 [ 0 11  1]
 [ 0  0  9]]
```

In [256…
```
cm = confusion_matrix(y_test, predictions)
```

```
sns.heatmap(cm, annot = True, fmt = "d", cmap ='Purples')
```

Out[256]:

```
<AxesSubplot:>
```