NAIVE BAYERS

Naive Bayes is a simple probabilistic classifier that predicts the category of an item based on the likelihood of its features, assuming each feature is independent of the others

- Multinomial Naive Bayes: Best for count-based features or text data.
- Bernoulli Naive Bayes: Best for binary features.
- Gaussian Naive Bayes: Best for continuous features with normal distribution.

In [1]:
```
import seaborn as sns
```

In [2]:
```
sns.get_dataset_names()
```

```
Out[2]: ['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'dowjones',
 'exercise',
 'flights',
 'fmri',
 'geyser',
 'glue',
 'healthexp',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'seaice',
 'taxis',
 'tips',
 'titanic',
 'anagrams',
 'anagrams',
 'anscombe',
 'anscombe',
 'attention',
 'attention',
 'brain_networks',
 'brain_networks',
 'car_crashes',
 'car_crashes',
 'diamonds',
 'diamonds',
 'dots',
 'dots',
 'dowjones',
 'dowjones',
 'exercise',
 'exercise',
 'flights',
 'flights',
```

```
'fmri',
'fmri',
'geyser',
'geyser',
'glue',
'glue',
'healthexp',
'healthexp',
'iris',
'iris',
'mpg',
'mpg',
'penguins',
'penguins',
'planets',
'planets',
'seaice',
'seaice',
'taxis',
'taxis',
'tips',
'tips',
'titanic',
'titanic',
'anagrams',
'anscombe',
'attention',
'brain_networks',
'car_crashes',
'diamonds',
'dots',
'dowjones',
'exercise',
'flights',
'fmri',
'geyser',
'glue',
'healthexp',
'iris',
'mpg',
'penguins',
'planets',
```

```
 'seaice',
 'taxis',
 'tips',
 'titanic']
```

In [3]:
```
data = sns.load_dataset("iris")
data
```

Out[3]:

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

150 rows × 5 columns

In [4]:
```
data.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 [5]:
```
data.describe()
```

Out[5]:

	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 [6]:
```
data.shape
```

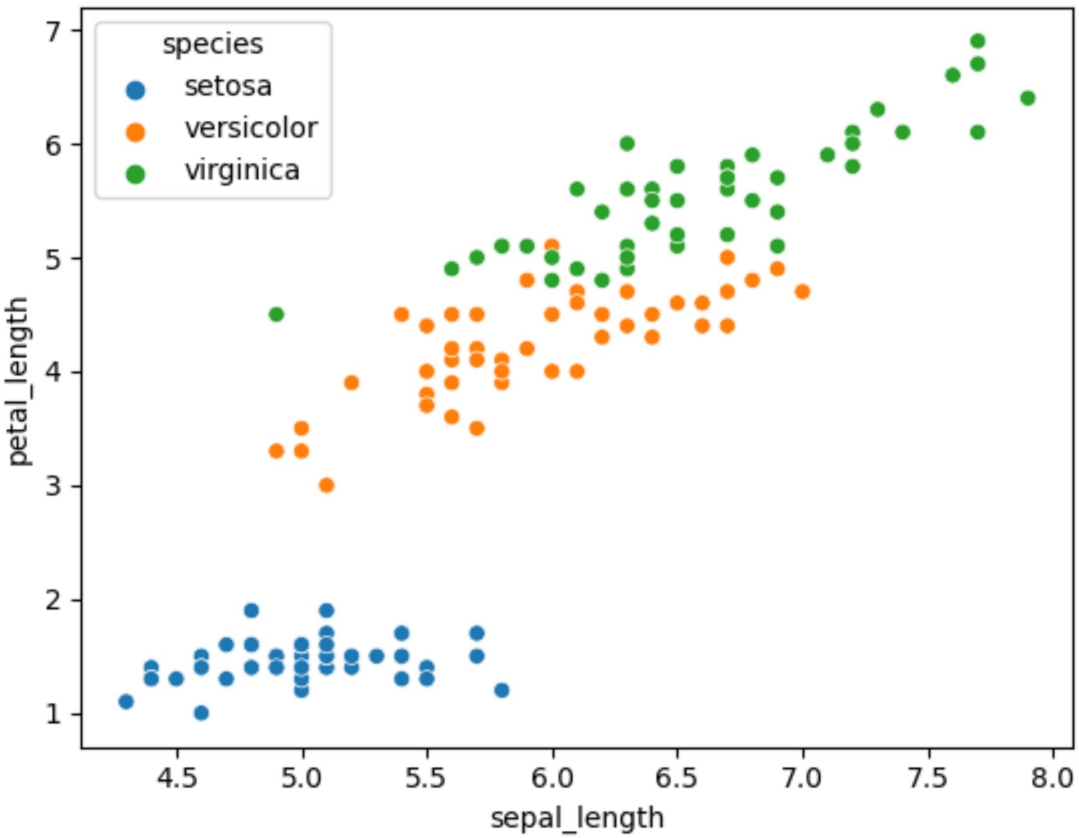
Out[6]: (150, 5)

In [7]:
```
data.isnull().sum()
```

Out[7]:
```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

In [8]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.scatterplot(x=data["sepal_length"],y=data["petal_length"],hue=data["species"])
plt.show()
```



In [9]:
```
from sklearn.model_selection import train_test_split
```

```
x = data.drop("species",axis=1) #Features
y = data["species"] # target
# Split the data into training and tesing sets
x_train, x_test, y_train, y_test =train_test_split(x,y)
```

In [10]:
```
x_train.shape
```

Out[10]: (112, 4)

In [11]:
```
y_train.shape
```

Out[11]: (112,)

In [12]:
```
x_test.shape
```

Out[12]: (38, 4)

In [13]:
```
y_test.shape
```

Out[13]: (38,)

In [14]:
```
# KNN

from sklearn.neighbors import KNeighborsClassifier

# create a KNN classifier
knn = KNeighborsClassifier(n_neighbors=3)

# fit the model
knn.fit(x_train , y_train)
```

Out[14]:
```
▾ KNeighborsClassifier

KNeighborsClassifier(n_neighbors=3)
```

In [15]:
```
# Predict on the test set
y_pred = knn.predict(x_test)

from sklearn.metrics import accuracy_score  , confusion_matrix , classification_report
```

```
# Calculate accuracy
accuracy = accuracy_score(y_test , y_pred)
print("Accuracy: ",accuracy)

# classification report
cr = classification_report(y_test ,y_pred)
print("Classification report: ",cr)

# confusion matrix
cm = confusion_matrix(y_test , y_pred)
print("Confusion Matrix : \n",cm)
```

```
Accuracy:  0.9473684210526315
Classification report:                precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       1.00      0.86      0.92        14
   virginica       0.82      1.00      0.90         9

    accuracy                           0.95        38
   macro avg       0.94      0.95      0.94        38
weighted avg       0.96      0.95      0.95        38

Confusion Matrix :
 [[15  0  0]
 [ 0 12  2]
 [ 0  0  9]]
```

In [16]:

```
from sklearn.naive_bayes import GaussianNB
```

In [17]:

```
# Create a Gaussian Naive Bayes model
model = GaussianNB()

# Train the model
model.fit(x_train, y_train)
```

Out[17]:

```
▾ GaussianNB

GaussianNB()
```

In [18]:

```
from sklearn.metrics import accuracy_score  , confusion_matrix , classification_report
```

In [19]:
```
# Make prediction
y_pred = model.predict(x_test)
```

In [20]:
```
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix :\n", cm)
```

```
Confusion Matrix :
 [[15  0  0]
 [ 0 12  2]
 [ 0  0  9]]
```

In [21]:
```
# Evaluate accuracy
accuracy = accuracy_score(y_test , y_pred)
print("Accuracy: ", accuracy)
```

```
Accuracy:  0.9473684210526315
```

In [22]:
```
cr= classification_report(y_test,y_pred)
print("Classification report :",cr)
```

```
Classification report :               precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       1.00      0.86      0.92        14
   virginica       0.82      1.00      0.90         9

    accuracy                           0.95        38
   macro avg       0.94      0.95      0.94        38
weighted avg       0.96      0.95      0.95        38
```

Conclusion

In this analysis, we compared K-Nearest Neighbors (KNN) and Gaussian Naive Bayes (GNB) classifiers using the Iris dataset.

1. **Data Visualization**: We visualized the relationship between sepal length and petal length, providing insights into species distribution.
2. **Model Training and Evaluation**:
   - **KNN**: Trained with (k = 3), we evaluated its accuracy, confusion matrix, and classification report.

- **GNB**: We also trained a GNB model and assessed its performance using the same metrics.

3. **Performance Insights**:

    - Both models were evaluated on accuracy and detailed metrics (precision, recall, F1-score).
    - The results showed how each model handled the classification of different iris species.

Key Takeaways:

- **Model Performance**: The choice of model matters; KNN may perform better with local patterns, while GNB is simpler and relies on feature independence.
- **Comprehensive Evaluation**: Using multiple metrics helps in understanding model effectiveness.
- **Continuous Improvement**: Exploring different models and hyperparameters can enhance results.

This analysis underscores the importance of model selection and thorough evaluation to achieve the best classification performance.