OASIS INFOBYTE INTERNSHIP

TASK-1

AUTHOR-KRIPA JOSE

IRIS FLOWER CLASSIFICATION

Importing libraries and data cleaning

In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import warnings
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
warnings.filterwarnings("ignore")
```

Import the iris dataset.

In [5]:
```
df=pd.read_csv("Iris.csv")
```

In [6]:
```
df
```

Out[6]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

Check the shape, data types, and summary statistics of the dataset

In [7]:
```
df.shape
```

Out[7]:
```
(150, 6)
```

In [8]:
```
df.dtypes
```

Out[8]:
```
Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

In [9]:
```
df.describe()
```

Out[9]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

Check for missing values and duplicate rows in the dataset.

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [11]:
```
df.duplicated().sum()
```

Out[11]: 0

Count the number of observations for each species in the dataset.

In [12]:
```
df['Species'].value_counts()
```

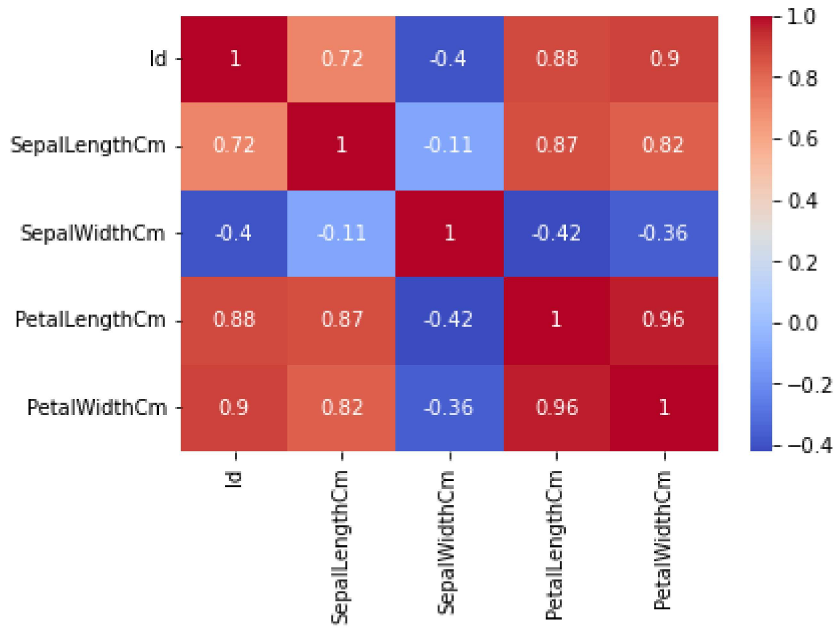
Out[12]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

Given dataset has 150 rows and 6 columns. The dataset do not have neither any missing values nor duplicated values. The target variable in this dataset is 'Species'. We have 3 categories in 'Species' column namely, 'Iris-setosa', 'Iris-versicolor' and 'Iris-virginica' distributed equally in the dataset.

Data Visualization

Create a heatmap of the correlation matrix to visualize the relationships between the features.

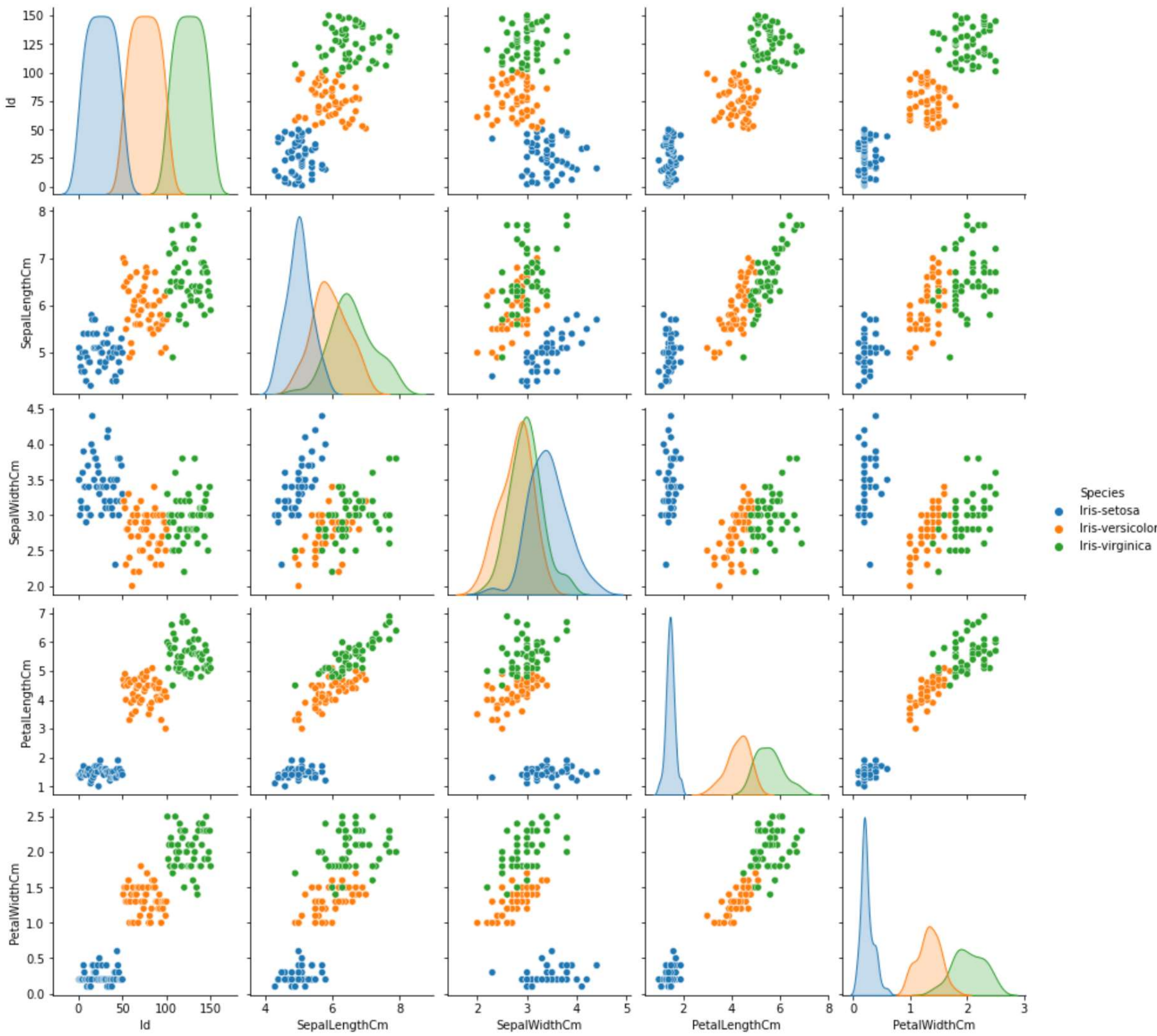
In [13]:
```
sns.heatmap(df.corr(),cmap='coolwarm',annot=True)
plt.show()
```



There is high positive correlation among 'SepalLengthCm','PetalLengthCm' and 'PetalWidthCm' which implies that if one of the dimension is high for a flower, most probably the other two dimensions will also be high. But we notice that 'SepalWidthCm' is negatively correlated with other 3 physical features. This shows that flowers with low sepal width will be having high value for the other 3 features.

Create a pairplot of the data to visualize the distribution of the features for each species.

In [14]:
```
sns.pairplot(data=df,hue='Species')
plt.show()
```



From the above pairplot we see clusters of each species for different feature combination. For most plots, the clusters are separated.

Model Building

Split the dataset into training and testing sets.

In [15]:
```
X=df.drop(['Id','Species'],axis=1)
X
```

Out[15]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [16]:
```
y=df['Species']
y
```

Out[16]:
```
0          Iris-setosa
1          Iris-setosa
2          Iris-setosa
3          Iris-setosa
4          Iris-setosa
              ...
145     Iris-virginica
146     Iris-virginica
147     Iris-virginica
148     Iris-virginica
149     Iris-virginica
Name: Species, Length: 150, dtype: object
```

In [17]:
```
# split the data into training and testing data using the train_test_split fun

X_train,X_test,y_train,y_test=train_test_split(X,y,train_size=0.7,random_state
```

In [18]:
```
X_train
```

Out[18]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
118	7.7	2.6	6.9	2.3
18	5.7	3.8	1.7	0.3
4	5.0	3.6	1.4	0.2
45	4.8	3.0	1.4	0.3
59	5.2	2.7	3.9	1.4
...	...	...	...	...
133	6.3	2.8	5.1	1.5
137	6.4	3.1	5.5	1.8
72	6.3	2.5	4.9	1.5
140	6.7	3.1	5.6	2.4
37	4.9	3.1	1.5	0.1

105 rows × 4 columns

In [19]:
```
y_train
```

Out[19]:
```
118     Iris-virginica
18          Iris-setosa
4           Iris-setosa
45          Iris-setosa
59     Iris-versicolor
            ...
133     Iris-virginica
137     Iris-virginica
72     Iris-versicolor
140     Iris-virginica
37          Iris-setosa
Name: Species, Length: 105, dtype: object
```

Logistic Regression

Here we are going to build our model using logistic regressor

In [20]:
```
log=LogisticRegression()
log.fit(X_train,y_train)
```

Out[20]:
```
LogisticRegression()
```

After training the model, we can use test data for prediction

In [21]:
```
y_pred=log.predict(X_test)
```

Evaluate the model using accuracy, precision, recall, and f1 score.

In [22]:
```
accuracy=accuracy_score(y_test,y_pred)
accuracy
```

Out[22]:
```
0.9777777777777777
```

In [23]:
```
print(classification_report(y_test,y_pred))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        14
Iris-versicolor       1.00      0.94      0.97        18
 Iris-virginica       0.93      1.00      0.96        13

       accuracy                           0.98        45
      macro avg       0.98      0.98      0.98        45
   weighted avg       0.98      0.98      0.98        45
```

Print the confusion matrix of the model.

In [24]:
```
print(confusion_matrix(y_test,y_pred))
```

```
[[14  0  0]
 [ 0 17  1]
 [ 0  0 13]]
```

Our model is 97.77% accurate and have high value for precision, recall and f1 score. The confusion matrix of our model is also good.

In [ ]:

In [ ]: