Iris Flower Classification

This a machine learning project. This project's main objective is to build a model to detects iris flowers correctly.

1. Importing all the libraries which we needed in this project.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
from scipy import stats

from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
```

2. Load the Iris.csv dataset in this nootbook and check all of its details.

In [2]:
```
iris = pd.read_csv('Iris.csv')
```

In [3]:
```
iris.head()
```

Out[3]:

	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

In [4]:
```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

2.1 Dropping the index column as we don't need of it.

In [5]:
```
iris.drop('Id',axis= 1, inplace =True)
```

In [6]:
```
iris['Species'].value_counts()
```

Out[6]:
```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
```

2.2 Finding statistical distribution and correlation of the data. It gives better idea of the data and how they are related?

In [7]:
```
iris.describe()
```

Out[7]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
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 [8]:
```
iris.corr(numeric_only = True)
```

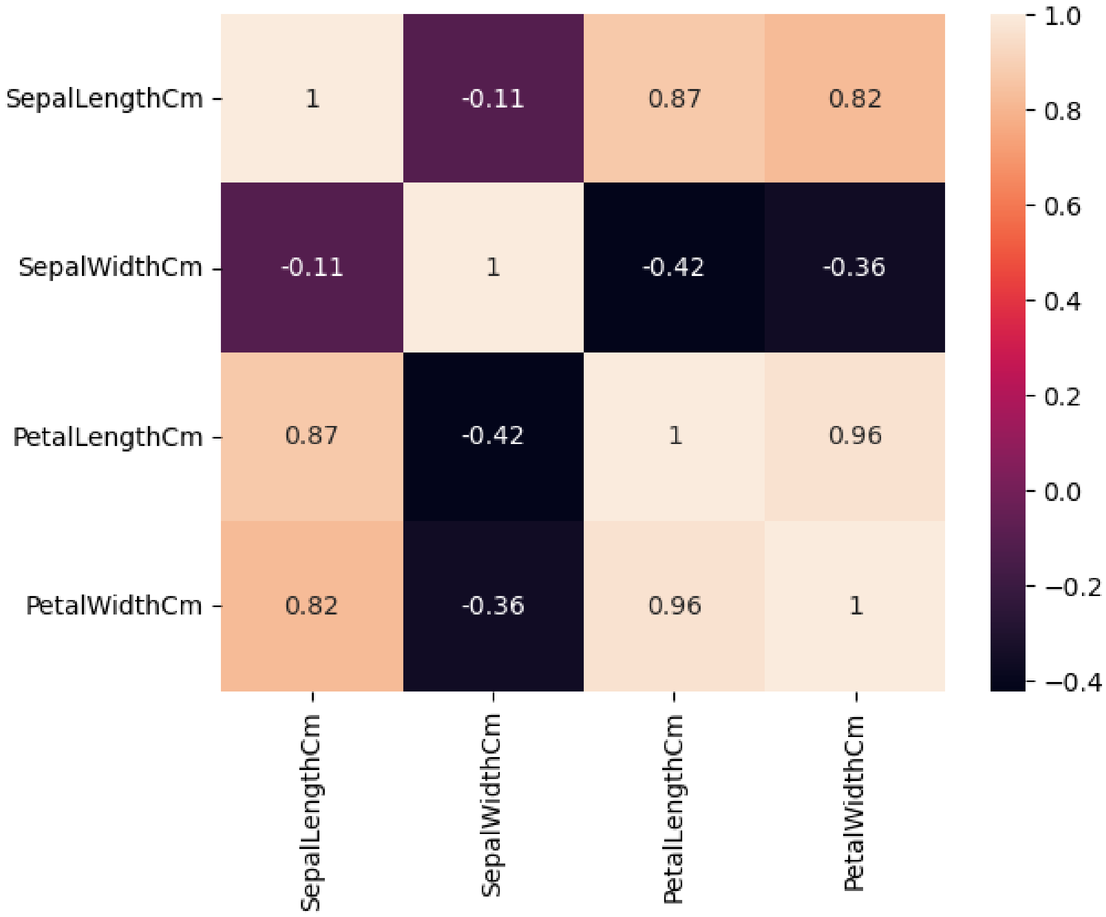
Out[8]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000

2.3 Visualize the correlation of the data to see clear picture of it.

In [9]:
```
sns.heatmap(iris.corr(numeric_only = True),annot =True)
```

Out[9]: <AxesSubplot:>

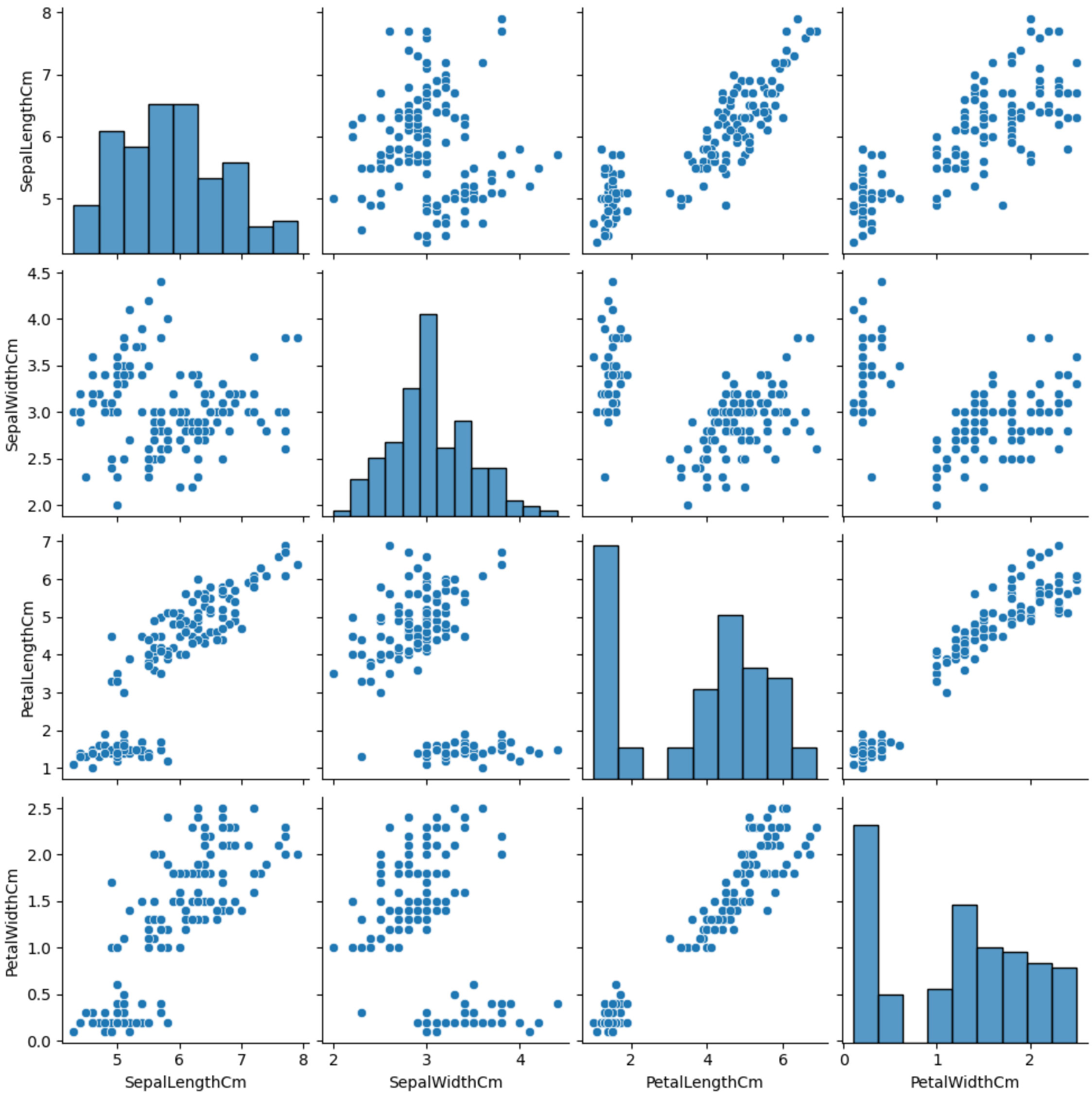


- From the above heat map we can say that except 'SepalWidthcm' all columns have a positive coorelation with each other.

In [10]:
```python
plt.figure(figsize= (0.1,0.1))
sns.pairplot(iris)
```

Out[10]:
```
<seaborn.axisgrid.PairGrid at 0x1b05f350cd0>
```

```
<Figure size 10x10 with 0 Axes>
```



3. Classification Model Making Process Starts.

3.1 Assigning the strings in the column 'Species' to 0, 1, 2 respectivly to Iris-setosa, Iris-versicolor,Iris-virginica.

In [11]:
```python
lr = LabelEncoder()
iris['Species'] = lr.fit_transform(iris['Species'])
```

- Iris-setosa : 0
- Iris-versicolor : 1
- Iris-virginica : 2

3.2 Preparaing the data for training and testing in Models

In [12]:
```
X = iris.drop('Species',axis= 1)
```

In [13]:
```
y = iris.Species
```

In [14]:
```
X_train, X_test, y_train, y_valid = train_test_split(X,y,random_state =42,t
```

In [15]:
```
scaler = StandardScaler()
X_train_k = scaler.fit_transform(X_train)
X_test_k = scaler.fit_transform(X_test)
print("Done!!!")
```

```
Done!!!
```

In [16]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_valid.shape)
```

```
(120, 4) (120,)
(30, 4) (30,)
```

3.3 Decision Tree Regressor Model.

In [17]:
```
tree = DecisionTreeRegressor()
```

In [18]:
```
tree.fit(X_train,y_train)
```

Out[18]:
```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

In [19]:
```
y_hat = tree.predict(X_test)
```

In [20]:
```
tree.score(X_train,y_train)*100
```

Out[20]:
```
100.0
```

In [21]:
```
tree.score(X_test,y_valid)*100
```

Out[21]:
```
100.0
```

In [22]:
```
mse_d = mean_squared_error(y_valid,y_hat)
rmse_d = np.sqrt(mse_d)
print("Root mean square error {:0.2f}".format(rmse_d))
print("R2_score Linear Regression {:0.2f}".format(r2_score(y_valid,y_hat)))
print("Mean Absoute Error {:0.2f}".format(mean_absolute_error(y_valid,y_hat
```

```
Root mean square error 0.00
R2_score Linear Regression 1.00
Mean Absoute Error 0.00
```

3.4 Random Forest Regressor Model.

In [23]:
```
Forest = RandomForestRegressor()
```

In [24]:
```
Forest.fit(X_train,y_train)
```

Out[24]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [25]:
```
Forest.score(X_train,y_train)*100
```

Out[25]:
```
99.0992720751134
```

In [26]:
```
y_fos = Forest.predict(X_test)
```

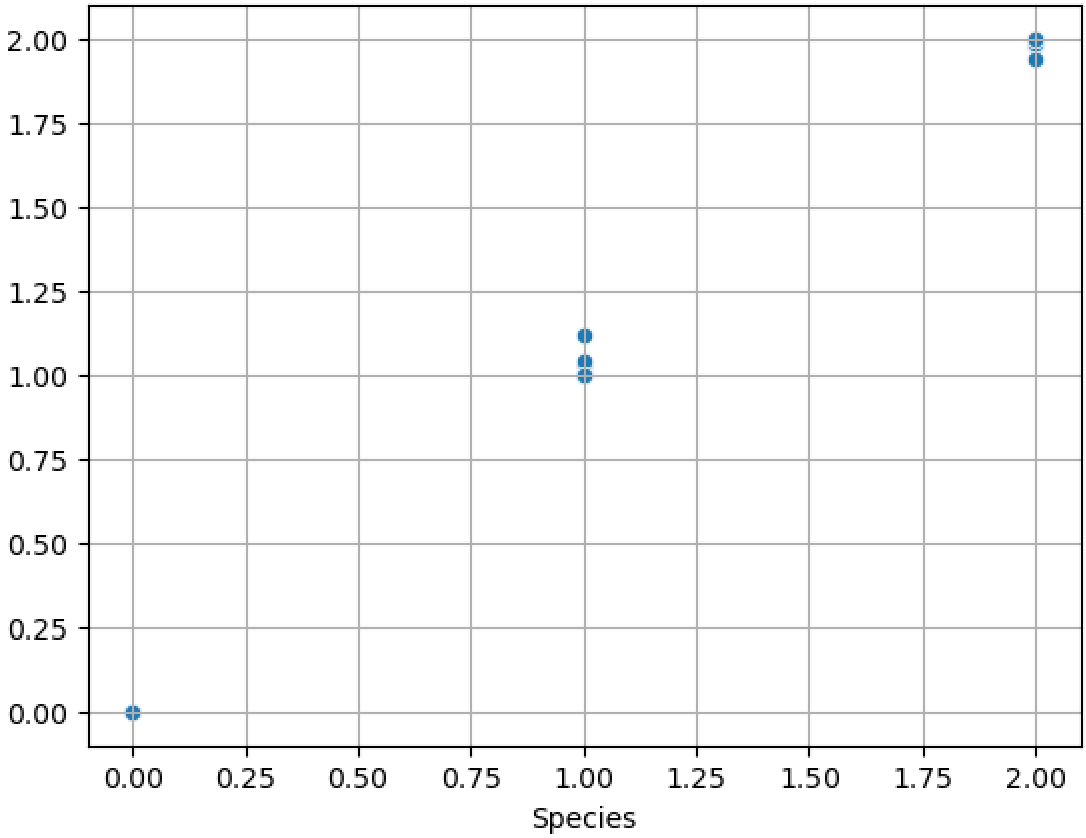
In [27]:
```
Forest.score(X_test,y_valid)*100
```

Out[27]:
```
99.89793322734499
```

In [28]:
```
mse_for = mean_squared_error(y_valid,y_fos)
rmse_for = np.sqrt(mse_for)
print("Root Mean Squared Error of Froest Model {:0.2f}".format(rmse_for))
print("R2 Score of Forest Model {:0.2f}".format(r2_score(y_valid,y_fos)))
print("Mean Absolute Error of Forest Model {:0.2f}".format(mean_absolute_er
```

```
Root Mean Squared Error of Froest Model 0.03
R2 Score of Forest Model 1.00
Mean Absolute Error of Forest Model 0.01
```

```
In [29]: sns.scatterplot(x=y_valid,y= y_fos)
         plt.grid()
```



3.5 Logistic Regression Model.

```
In [30]: lg = LogisticRegression(max_iter=1000)
```

```
In [31]: lg.fit(X_train,y_train)
```

```
Out[31]: ▾ LogisticRegression
         LogisticRegression(max_iter=1000)
```

```
In [32]: lg.score(X_train,y_train)*100
```

```
Out[32]: 97.5
```

```
In [33]: y_lg = lg.predict(X_test)
```

```
In [34]: lg.score(X_test,y_valid)*100
```

```
Out[34]: 100.0
```

In [38]:
```
mse_lg = mean_squared_error(y_valid,y_lg)
rmse_lg = np.sqrt(mse_lg)
print("Root mean squared error of logistic Regression {:0.2f}".format(rmse_
print("R2 score of logistic Regression {:0.2f}".format(r2_score(y_valid,y_l
print("Mean absolute error of logistic Regression {:0.2f}".format(mean_abso
```

```
Root mean squared error of logistic Regression 0.00
R2 score of logistic Regression 1.00
Mean absolute error of logistic Regression 0.00
```

3.6 KNeighbors Classifier Model.

In [39]:
```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train_k, y_train)
y_knn = knn.predict(X_test_k)
print("K-Neighbors Model score in train set :",knn.score(X_train_k,y_train)
print("K-Neighbors Model score in test set :",knn.score(X_test_k,y_valid)*1

print("Confusion Matrix :\n",confusion_matrix(y_valid, y_knn))
print("Classification Report :\n",classification_report(y_valid, y_knn))
```

```
K-Neighbors Model score in train set : 100.0
K-Neighbors Model score in test set : 96.66666666666667
Confusion Matrix :
 [[10  0  0]
 [ 0  8  1]
 [ 0  0 11]]
Classification Report :
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.89      0.94         9
           2       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30
```

In [40]:
```
mse_knn = mean_squared_error(y_valid,y_knn)
rmse_knn = np.sqrt(mse_knn)
print("Root mean squared error of K-Neighbors Model {:0.2f}".format(rmse_kr
print("R2 score of K-Neighbors Model {:0.2f}".format(r2_score(y_valid,y_knr
print("Mean absolute error of K-Neighbors Model {:0.2f}".format(mean_absolu
```

```
Root mean squared error of K-Neighbors Model 0.18
R2 score of K-Neighbors Model 0.95
Mean absolute error of K-Neighbors Model 0.03
```

Thank You!!!