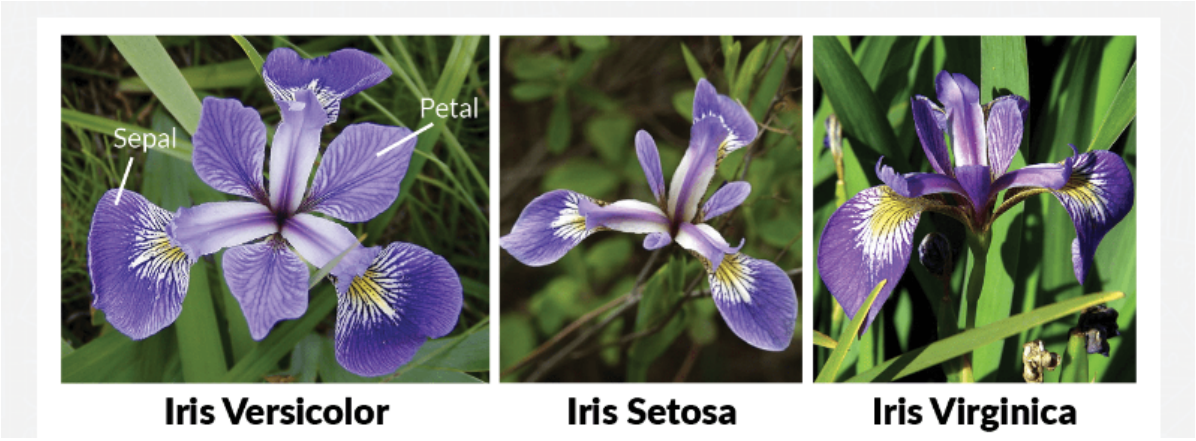
INTRODUCTION TO DATA SET

The Iris Dataset contains four features (length and width of sepals and petals) of 150 samples of three species of Iris ie classes (Iris setosa, Iris virginica and Iris versicolor).



To featch data from the given data set file.

In [1]:
```
import numpy as np
import pandas as pd
```

To load the data

In [20]:
```
data=pd.read_csv("D:\\gitsessions\\Iris.CSV")
data
```

Out[20]:

	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

In [6]:
```
data.head()  # to get the top 5 rows
```

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

01) To Segregate features and classes

In [13]:
```
features=list(data.columns)
features
```

Out[13]:
```
['Id',
 'SepalLengthCm',
 'SepalWidthCm',
 'PetalLengthCm',
 'PetalWidthCm',
 'Species']
```

In [19]:
```
classes=list(data.Species.unique())
classes
```

Out[19]:
```
['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
```

02) Spliting the data into train and test data in the ratio 70:30

we have to shuffle the dataset to assure an even distribution of classes when splitting the dataset into training and test set.

In [21]:
```
data = data.sample(frac=1, random_state=42)
data.set_index("Id", inplace=True)
data.head()
```

Out[21]:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
74	6.1	2.8	4.7	1.2	Iris-versicolor
19	5.7	3.8	1.7	0.3	Iris-setosa
119	7.7	2.6	6.9	2.3	Iris-virginica
79	6.0	2.9	4.5	1.5	Iris-versicolor
77	6.8	2.8	4.8	1.4	Iris-versicolor

In [26]:
```
train_dataset=data[:106]
test_dataset=data[106:]
```

In [31]:
```
train_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 106 entries, 74 to 7
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   SepalLengthCm  106 non-null    float64
 1   SepalWidthCm   106 non-null    float64
 2   PetalLengthCm  106 non-null    float64
 3   PetalWidthCm   106 non-null    float64
 4   Species        106 non-null    object 
dtypes: float64(4), object(1)
memory usage: 5.0+ KB
```

In [33]:
```
train_dataset.describe()
```

Out[33]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	106.000000	106.000000	106.000000	106.000000
mean	5.765094	3.078302	3.581132	1.124528
std	0.842111	0.459193	1.817854	0.778865
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.500000	0.225000
50%	5.700000	3.000000	4.150000	1.300000
75%	6.300000	3.400000	5.075000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [34]:
```
test_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 44 entries, 113 to 103
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   SepalLengthCm  44 non-null     float64
 1   SepalWidthCm   44 non-null     float64
 2   PetalLengthCm  44 non-null     float64
 3   PetalWidthCm   44 non-null     float64
 4   Species        44 non-null     object 
dtypes: float64(4), object(1)
memory usage: 2.1+ KB
```

In [35]:
```
test_dataset.describe()
```

Out[35]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	44.000000	44.000000	44.000000	44.000000
mean	6.031818	2.995455	4.186364	1.377273
std	0.770011	0.362776	1.566301	0.700453
min	4.500000	2.300000	1.200000	0.100000
25%	5.475000	2.800000	3.800000	1.150000

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
50%	6.100000	3.000000	4.600000	1.400000
75%	6.625000	3.125000	5.425000	1.825000
max	7.400000	4.000000	6.300000	2.500000

03) Train the "RandomForestClassifier" and "LogisticRegression"

DATA PREPRATION

First we have to save our features and our target variable in separate dataframes for both the training and the test set.

In [40]:
```
X_train = train_dataset.drop("Species", axis=1)
y_train = train_dataset["Species"].copy()

X_test = test_dataset.drop("Species", axis=1)
y_test = test_dataset["Species"].copy()
```

In [44]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

my_pipeline = Pipeline(steps=[
    ("scaler", StandardScaler()),
    ("impute", SimpleImputer(strategy="mean"))
])
```

In [45]:
```
from sklearn.compose import ColumnTransformer

full_pipeline = ColumnTransformer([
    ("full", my_pipeline, X_train.columns)
])
```

In [46]:
```
X_train_prepared = full_pipeline.fit_transform(X_train)
```

"RANDOM FOREST CLASSIFIER"

In [59]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

forest_clf = RandomForestClassifier()

scores_forest = cross_val_score(forest_clf, X_train_prepared, y_train, cv=4, scoring
```

In [60]:
```
print(scores_forest)
print("mean: ", scores_forest.mean())
print("Std: ", scores_forest.std())
```

```
[1.         0.96296296 0.96153846 0.88461538]
```

```
mean:  0.9522792022792023
Std:  0.04199864099089907
```

"LOGISTIC REGRESSION"

In [61]:

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()

scores_log = cross_val_score(log_reg, X_train_prepared, y_train, cv=4, scoring="accu
print(scores_log)
print("mean: ", scores_log.mean())
print("Std: ", scores_log.std())
```

```
[1.         0.96296296 0.96153846 0.84615385]
mean:  0.9426638176638177
Std:  0.05781418486628931
```

04) CLEARLY FROM THE ABOVE RESULT "RANDOM FOREST CLASSIFIER" HAVE MORE ACCURACY THAN "LOGISTIC REGRESSION"

RANDOM FOREST CLASSIFIER

[1. 0.96296296 0.96153846 0.88461538] mean: 0.9522792022792023 Std: 0.04199864099089907

LOGISTIC REGRESSION

[1. 0.96296296 0.96153846 0.84615385] mean: 0.9426638176638177 Std: 0.05781418486628931

In [ ]:

In [ ]: