Bhumi Reddy Chenna Kesava Reddy

1)Iris Flowers Classification ML Project :

import Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Import Dataset

In [2]:

```
df=pd.read_csv('iris data.csv')
df
```

Out[2]:

	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

Preprocessing the data

In [3]:

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

Out[3]:

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

In [4]:

```
df.isna()
```

Out[4]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
145	False	False	False	False	False
146	False	False	False	False	False
147	False	False	False	False	False
148	False	False	False	False	False
149	False	False	False	False	False

150 rows × 5 columns

In [5]:

```
df.shape
```

Out[5]:

```
(150, 5)
```

In [6]:

```
df.isna().sum()
```

Out[6]:

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

In [7]:

```
df.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 [8]:

```
df.describe()
```

Out[8]:

	sepal_length	sepal_width	petal_length	petal_width
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 [9]:

```
len(df)
```

Out[9]:

```
150
```

df.head()

In [11]:

```
x=df.iloc[:,:-1].values
y=df.iloc[:,-1:].values
```

Encoding the Categorical data of Dependent Variable

In [12]:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct=ColumnTransformer(transformers=[('encoder',OneHotEncoder(),[0])],remainder='passthrou
y=ct.fit_transform(y)
y
```

Out[12]:

```
array([[1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
```

Splitting the data into training and test data

In [13]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=21)
```

Random Forest Classification on Training data

In [14]:

```
from sklearn.ensemble import RandomForestClassifier
classifier=RandomForestClassifier(n_estimators=100)
classifier.fit(x_train,y_train)
```

Out[14]:

```
RandomForestClassifier()
```

In [63]:

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

In [61]:

```
print(np.argmax(classifier.predict([[6.7,3.0,5.2,2.3]])))
```

```
2
```

Predicting the Test Results

In [71]:

```
y_pred=classifier.predict(x_test)
#y_pred = np.argmax(y_pred, axis=1)
print(y_pred,y_test)
```

```
[[0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 0. 1.]] [1 0 0 0 1 1 0 2 0 0 1 1 2 2 0 1 2 1 0 2 2 1 1 1 0 1 0 0 1 2
0 2 1 0 2 1 1
 2 0 2 2 2 2 1 2]
```

In [76]:

```
categorical_data = np.argmax(y_pred, axis=1)
text_data = le.inverse_transform(categorical_data)
print(text_data)
```

```
[1 0 0 0 1 1 0 2 0 0 1 1 2 2 0 2 2 1 0 2 2 1 2 1 0 1 0 0 1 2 0 2 2 0 2 1 1
 2 0 2 2 2 2 1 2]
```

Confusion Matrix

In [57]:

```
from sklearn.metrics import r2_score,accuracy_score,confusion_matrix
cm=confusion_matrix(y_pred,y_test)
print(cm)
```

```
[[14  0  0]
 [ 0 13  0]
 [ 0  3 15]]
```

Accuracy

In [58]:

```
accuracy=accuracy_score(y_pred,y_test)
print(accuracy)
```

```
0.9333333333333333
```

Accuracy 93.3%

Thank You