In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
df=pd.read_csv('Zoo.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	animal name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs	tail
0	aardvark	1	0	0	1	0	0	1	1	1	1	0	0	4	0
1	antelope	1	0	0	1	0	0	0	1	1	1	0	0	4	1
2	bass	0	0	1	0	0	1	1	1	1	0	0	1	0	1
3	bear	1	0	0	1	0	0	1	1	1	1	0	0	4	0
4	boar	1	0	0	1	0	0	1	1	1	1	0	0	4	1

In [5]:

```
df.shape
```

Out[5]:

```
(101, 18)
```

In [7]:

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

Out[7]:

```
animal name    0
hair           0
feathers       0
eggs           0
milk           0
airborne       0
aquatic        0
predator       0
toothed        0
backbone       0
breathes       0
venomous       0
fins           0
legs           0
tail           0
domestic       0
catsize        0
type           0
dtype: int64
```

cHECK dUPLICATE vALUES

In [4]:

```
df.duplicated().sum()
```

Out[4]:

0

Checking for Outliers and removing it

In [8]:

```
Q1=df.quantile(0.25)
Q3=df.quantile(0.75)
IQR=Q3-Q1
whisker_width=1.5
in_outliers = df[(df< Q1 - whisker_width*IQR) | (df> Q3 + whisker_width*IQR)]
in_outliers
```

```
<ipython-input-8-d2aeeaa1e73e>:5: FutureWarning: Automatic reindexing on DataFrame vs Ser
ies comparisons is deprecated and will raise ValueError in a future version.  Do `left, r
ight = left.align(right, axis=1, copy=False)` before e.g. `left == right`
  in_outliers = df[(df< Q1 - whisker_width*IQR) | (df> Q3 + whisker_width*IQR)]
<ipython-input-8-d2aeeaa1e73e>:5: FutureWarning: Automatic reindexing on DataFrame vs Ser
ies comparisons is deprecated and will raise ValueError in a future version.  Do `left, r
ight = left.align(right, axis=1, copy=False)` before e.g. `left == right`
  in_outliers = df[(df< Q1 - whisker_width*IQR) | (df> Q3 + whisker_width*IQR)]
```

Out[8]:

	animal name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs	ta
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	1.0	NaN	Na
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
96	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
97	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	0.0	NaN	1.0	NaN	NaN	Na
98	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
99	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	NaN	NaN	NaN	Na
100	NaN	NaN	1.0	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

101 rows × 18 columns

In [9]:

```
print((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR)))
```

```
     airborne  animal name  aquatic  backbone  breathes  catsize  domestic  \
0       False        False    False     False     False    False     False
1       False        False    False     False     False    False     False
2       False        False    False     False      True    False     False
3       False        False    False     False     False    False     False
4       False        False    False     False     False    False     False
..        ...          ...      ...       ...       ...      ...       ...
96      False        False    False     False     False    False     False
97       True        False    False      True     False    False     False
98      False        False    False     False     False    False     False
99      False        False    False      True     False    False     False
100      True        False    False     False     False    False     False

      eggs  feathers   fins   hair   legs   milk  predator   tail  toothed  \
0    False     False  False  False  False  False     False  False    False
1    False     False  False  False  False  False     False  False    False
```

```
2    False     False   True  False  False  False     False  False    False
3    False     False  False  False  False  False     False  False    False
4    False     False  False  False  False  False     False  False    False
..     ...       ...    ...    ...    ...    ...       ...    ...      ...
96   False     False  False  False  False  False     False  False    False
97   False     False  False  False  False  False     False  False    False
98   False     False  False  False  False  False     False  False    False
99   False     False  False  False  False  False     False  False    False
100  False      True  False  False  False  False     False  False    False

      type  venomous
0    False     False
1    False     False
2    False     False
3    False     False
4    False     False
..     ...       ...
96   False     False
97   False      True
98   False     False
99   False     False
100  False     False

[101 rows x 18 columns]
```

```
<ipython-input-9-4ac38365c18c>:1: FutureWarning: Automatic reindexing on DataFrame vs Ser
ies comparisons is deprecated and will raise ValueError in a future version.  Do `left, r
ight = left.align(right, axis=1, copy=False)` before e.g. `left == right`
  print((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR)))
<ipython-input-9-4ac38365c18c>:1: FutureWarning: Automatic reindexing on DataFrame vs Ser
ies comparisons is deprecated and will raise ValueError in a future version.  Do `left, r
ight = left.align(right, axis=1, copy=False)` before e.g. `left == right`
  print((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR)))
```

In [10]:

```
df1 = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
df1
```

```
<ipython-input-10-2078feffebbb>:1: FutureWarning: Automatic reindexing on DataFrame vs Se
ries comparisons is deprecated and will raise ValueError in a future version.  Do `left,
right = left.align(right, axis=1, copy=False)` before e.g. `left == right`
  df1 = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
<ipython-input-10-2078feffebbb>:1: FutureWarning: Automatic reindexing on DataFrame vs Se
ries comparisons is deprecated and will raise ValueError in a future version.  Do `left,
right = left.align(right, axis=1, copy=False)` before e.g. `left == right`
  df1 = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
```

Out[10]:

	animal name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs	ta
0	aardvark	1	0	0	1	0	0	1	1	1	1	0	0	4	
1	antelope	1	0	0	1	0	0	0	1	1	1	0	0	4	
3	bear	1	0	0	1	0	0	1	1	1	1	0	0	4	
4	boar	1	0	0	1	0	0	1	1	1	1	0	0	4	
5	buffalo	1	0	0	1	0	0	0	1	1	1	0	0	4	
10	cheetah	1	0	0	1	0	0	1	1	1	1	0	0	4	
17	deer	1	0	0	1	0	0	0	1	1	1	0	0	4	
22	elephant	1	0	0	1	0	0	0	1	1	1	0	0	4	
25	frog	0	0	1	0	0	1	1	1	1	1	0	0	4	
28	giraffe	1	0	0	1	0	0	0	1	1	1	0	0	4	
32	gorilla	1	0	0	1	0	0	0	1	1	1	0	0	2	
36	hare	1	0	0	1	0	0	0	1	1	1	0	0	4	

	animal name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs	ta
44	leopard	1	0	0	1	0	0	1	1	1	1	0	0	4	
45	lion	1	0	0	1	0	0	1	1	1	1	0	0	4	
47	lynx	1	0	0	1	0	0	1	1	1	1	0	0	4	
48	mink	1	0	0	1	0	1	1	1	1	1	0	0	4	
49	mole	1	0	0	1	0	0	1	1	1	1	0	0	4	
50	mongoose	1	0	0	1	0	0	1	1	1	1	0	0	4	
52	newt	0	0	1	0	0	1	1	1	1	1	0	0	4	
54	opossum	1	0	0	1	0	0	1	1	1	1	0	0	4	
55	oryx	1	0	0	1	0	0	0	1	1	1	0	0	4	
63	platypus	1	0	1	1	0	1	1	0	1	1	0	0	4	
64	polecat	1	0	0	1	0	0	1	1	1	1	0	0	4	
67	puma	1	0	0	1	0	0	1	1	1	1	0	0	4	
69	raccoon	1	0	0	1	0	0	1	1	1	1	0	0	4	
80	slowworm	0	0	1	0	0	0	1	1	1	1	0	0	0	
84	squirrel	1	0	0	1	0	0	0	1	1	1	0	0	2	
89	toad	0	0	1	0	0	1	0	1	1	1	0	0	4	
90	tortoise	0	0	1	0	0	0	0	0	1	1	0	0	4	
91	tuatara	0	0	1	0	0	0	1	1	1	1	0	0	4	
94	vole	1	0	0	1	0	0	0	1	1	1	0	0	4	
96	wallaby	1	0	0	1	0	0	0	1	1	1	0	0	2	
98	wolf	1	0	0	1	0	0	1	1	1	1	0	0	4	

In [11]:

```
df1.shape
```

Out[11]:

```
(33, 18)
```

Splitting Dataset into independent and Dependent Variables

In [24]:

```
x=df1.drop('type',axis=1)
y=df1.iloc[:,-1].values
```

In [25]:

```
x=x.drop('animal name',axis=1)
```

In [27]:

```
x=x.values
```

In [28]:

```
x
```

Out[28]:

```
array([[1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 4, 0, 0, 1],
       [1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 4, 1, 0, 1],
       [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 4, 0, 0, 1],
       [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 4, 1, 0, 1],
       [1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 4, 1, 0, 1],
```

```
       [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 4, 1, 0, 1],
       [1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 4, 1, 0, 1],
       [1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 4, 1, 0, 1],
       [0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 4, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 4, 1, 0, 1],
       [1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 2, 0, 0, 1],
       [1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 4, 1, 0, 0],
       [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 4, 1, 0, 1],
       [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 4, 1, 0, 1],
       [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 4, 1, 0, 1],
       [1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 4, 1, 0, 1],
       [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 4, 1, 0, 0],
       [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 4, 1, 0, 1],
       [0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 4, 1, 0, 0],
       [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 4, 1, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 4, 1, 0, 1],
       [1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 4, 1, 0, 1],
       [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 4, 1, 0, 1],
       [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 4, 1, 0, 1],
       [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 4, 1, 0, 1],
       [0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 2, 1, 0, 0],
       [0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 4, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 4, 1, 0, 1],
       [0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 4, 1, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 4, 1, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 2, 1, 0, 1],
       [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 4, 1, 0, 1]], dtype=int64)
```

In [15]:

```
y
```

Out[15]:

```
array(['Species1', 'Species1', 'Species1', 'Species1', 'Species1',
       'Species1', 'Species1', 'Species1', 'species5', 'Species1',
       'Species1', 'Species1', 'Species1', 'Species1', 'Species1',
       'Species1', 'Species1', 'Species1', 'species5', 'Species1',
       'Species1', 'Species1', 'Species1', 'Species1', 'Species1',
       'species3', 'species1', 'species5', 'species3', 'species3',
       'Species1', 'Species1', 'Species1'], dtype=object)
```

Splitting Data into Train-Test

In [59]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0
)
```

Applying Feature Scaling

In [60]:

```
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

Naive Bayers Classifier

In [61]:

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

In [62]:

```
gb=GaussianNB()
gb.fit(x_train,y_train)
```

Out[62]:

```
GaussianNB()
```

In [63]:

```
y_pred=gb.predict(x_test)
```

In [64]:

```
from sklearn.metrics import confusion_matrix
```

In [65]:

```
cb=confusion_matrix(y_test,y_pred)
```

In [66]:

```
cb
```

Out[66]:

```
array([[7, 0, 0],
       [1, 0, 0],
       [1, 0, 0]], dtype=int64)
```

In [67]:

```
from sklearn.metrics import accuracy_score
y_predicted = gb.predict(x_test)
accuracy_score(y_test, y_predicted)
```

Out[67]:

```
0.7777777777777778
```

Decision Tree

In [68]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [69]:

```
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [70]:

```
classifier.fit(x_train, y_train)
```

Out[70]:

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [71]:

```
y_pred2= classifier.predict(x_test)
```

In [72]:

```
c=confusion_matrix(y_test,y_pred2)
```

In [73]:

```
c
```

Out[73]:

```
array([[7, 0, 0],
       [1, 0, 0],
       [0, 0, 1]], dtype=int64)
```

In [74]:

```
y_predicted2 = classifier.predict(x_test)
accuracy_score(y_test, y_predicted2)
```

Out[74]:

```
0.8888888888888888
```

Random Forest

In [75]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [76]:

```
classifier3=RandomForestClassifier(n_estimators= 10, criterion="entropy")
```

In [77]:

```
classifier3.fit(x_train, y_train)
```

Out[77]:

```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

In [78]:

```
y_pred3=classifier3.predict(x_test)
```

In [79]:

```
cm3=confusion_matrix(y_test,y_pred3)
```

In [80]:

```
cm3
```

Out[80]:

```
array([[7, 0, 0],
       [1, 0, 0],
       [0, 0, 1]], dtype=int64)
```

In [81]:

```
y_predicted3 = classifier3.predict(x_test)
accuracy_score(y_test, y_predicted3)
```

Out[81]:

```
0.8888888888888888
```

K-Nearest Neighbour

In [82]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [83]:

```
classifier4= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
classifier4.fit(x_train,y_train)
```

Out[83]:

```
KNeighborsClassifier()
```

In [86]:

```
y_pred4=classifier4.predict(x_test)
```

In [87]:

```
cm4= confusion_matrix(y_test, y_pred4)
```

In [89]:

```
cm4
```

Out[89]:

```
array([[7, 0, 0],
       [1, 0, 0],
       [1, 0, 0]], dtype=int64)
```

In [90]:

```
y_predicted4 = classifier4.predict(x_test)
accuracy_score(y_test, y_predicted4)
```

Out[90]:

```
0.7777777777777778
```

Inference

The Overall Accuracy is 41.5 (88+78+88+78)/4

From the comparision we can find that the Random forest Algorithm and Decision Tree Algorithm gives 88% accuracy whereas in KNN and Naive Bayers gives 78% accuracy