KNN Q2 (Zoo)

Implement a KNN model to classify the animals in to categorie

1. Import Libs

In [1]:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
```

2. Import Data

In [2]:

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

Out[2]:

	animal name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	brea
0	aardvark	1	0	0	1	0	0	1	1	1	
1	antelope	1	0	0	1	0	0	0	1	1	
2	bass	0	0	1	0	0	1	1	1	1	
3	bear	1	0	0	1	0	0	1	1	1	
4	boar	1	0	0	1	0	0	1	1	1	
...	...	...	...	...	...	...	...	...	...	...	
96	wallaby	1	0	0	1	0	0	0	1	1	
97	wasp	1	0	1	0	1	0	0	0	0	
98	wolf	1	0	0	1	0	0	1	1	1	
99	worm	0	0	1	0	0	0	0	0	0	
100	wren	0	1	1	0	1	0	0	0	1	

101 rows × 18 columns

3. EDA

In [3]:

```
zoo.describe()
```

Out[3]:

	hair	feathers	eggs	milk	airborne	aquatic	predator	
count	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000	10
mean	0.425743	0.198020	0.584158	0.405941	0.237624	0.356436	0.554455	
std	0.496921	0.400495	0.495325	0.493522	0.427750	0.481335	0.499505	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	
75%	1.000000	0.000000	1.000000	1.000000	0.000000	1.000000	1.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

In [4]:

```
zoo.isna().sum()
```

Out[4]:

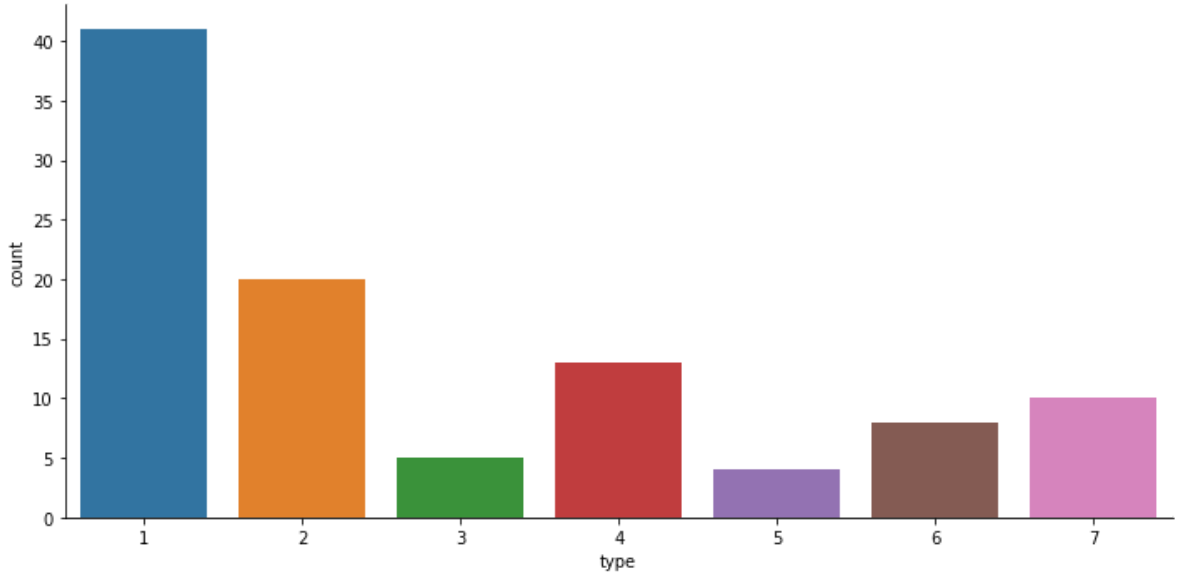
```
animal name    0
hair           0
feathers       0
eggs           0
milk           0
airborne       0
aquatic        0
predator       0
toothed        0
backbone       0
breathes       0
venomous       0
fins           0
legs           0
tail           0
domestic       0
catsize        0
type   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

In [5]:

```
sns.catplot('type',data=zoo,height = 5,kind="count",aspect=2)
```

Out[5]:

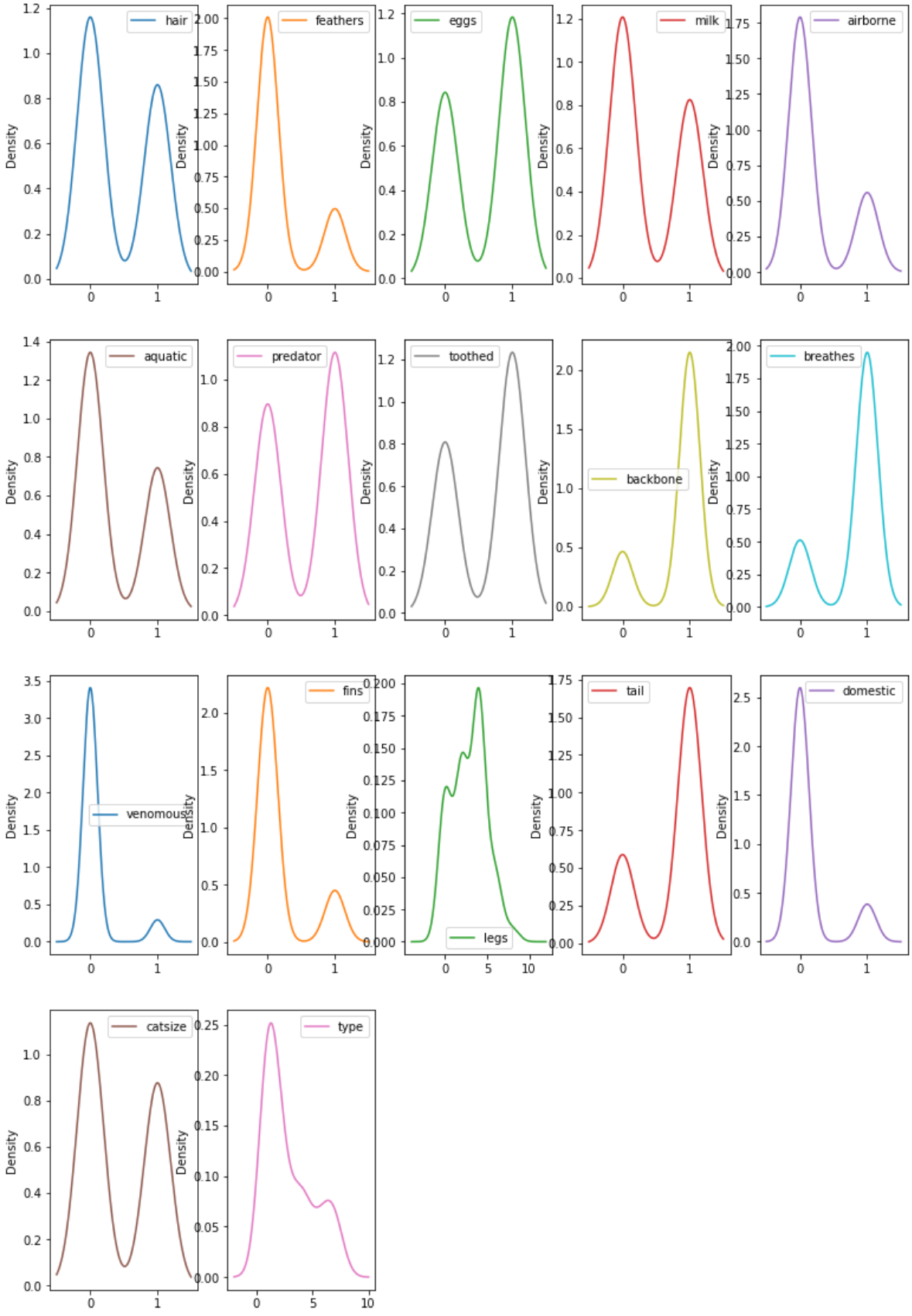
```
<seaborn.axisgrid.FacetGrid at 0x1bd8739f100>
```



As per the graph, highest number of animals available in Zoo are Type 1 followed by 2, 4 and 7 respectivel

In [6]:

```
zoo.plot(kind='density', subplots=True, layout=(4,5), figsize=(13,20),sharex=False)
plt.show()
```



4. Model Building

In [7]:

```
X = zoo.iloc[:,1:17]
y = zoo['type']
```

In [8]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,random_state= 12,strati
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

Out[8]:

```
((80, 16), (80,), (21, 16), (21,))
```

Finding optimal number of K

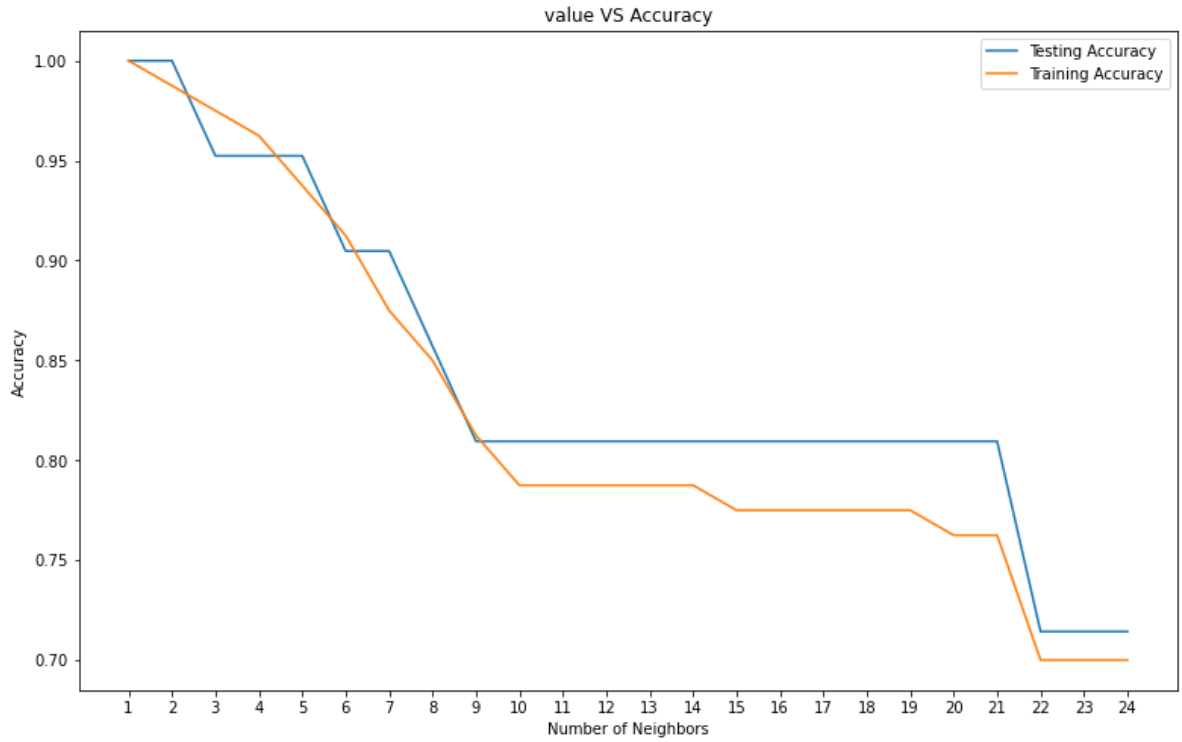
In [9]:

```
k_values = list(range(1,25))
train_accuracy = []
test_accuracy = []

for i, k in enumerate(k_values):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train,y_train)
    train_accuracy.append(knn.score(X_train, y_train))
    test_accuracy.append(knn.score(X_test, y_test))
```

In [10]:

```
plt.figure(figsize=[13,8])
plt.plot(k_values, test_accuracy, label = 'Testing Accuracy')
plt.plot(k_values, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.title('value VS Accuracy')
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.show()
```



as we can see k =4 has most accurate result ,so we'll go with that

5. KNN

Generating a Model with K = 4

Model Training without STANDARDIZATION

In [11]:

```
knn_model = KNeighborsClassifier(n_neighbors=4)
knn_model.fit(X_train,y_train)
y_pred = knn_model.predict(X_test)
print("Accuracy score: ", round(accuracy_score(y_test,y_pred),4))
```

```
Accuracy score:  0.9524
```

Model Training with STANDARDIZATION

In [12]:

```
scaler   = StandardScaler()
scaled_X = scaler.fit_transform(X)
```

In [13]:

```
pd.DataFrame(scaled_X)
```

Out[13]:

	0	1	2	3	4	5	6	7	
0	1.161395	-0.496904	-1.185227	1.209717	-0.558291	-0.744208	0.896421	0.809776	0.4
1	1.161395	-0.496904	-1.185227	1.209717	-0.558291	-0.744208	-1.115547	0.809776	0.4
2	-0.861034	-0.496904	0.843721	-0.826640	-0.558291	1.343710	0.896421	0.809776	0.4
3	1.161395	-0.496904	-1.185227	1.209717	-0.558291	-0.744208	0.896421	0.809776	0.4
4	1.161395	-0.496904	-1.185227	1.209717	-0.558291	-0.744208	0.896421	0.809776	0.4
...	...	...	...	...	...	...	...	...	
96	1.161395	-0.496904	-1.185227	1.209717	-0.558291	-0.744208	-1.115547	0.809776	0.4
97	1.161395	-0.496904	0.843721	-0.826640	1.791182	-0.744208	-1.115547	-1.234909	-2.1
98	1.161395	-0.496904	-1.185227	1.209717	-0.558291	-0.744208	0.896421	0.809776	0.4
99	-0.861034	-0.496904	0.843721	-0.826640	-0.558291	-0.744208	-1.115547	-1.234909	-2.1
100	-0.861034	2.012461	0.843721	-0.826640	1.791182	-0.744208	-1.115547	-1.234909	0.4

101 rows × 16 columns

In [14]:

```
X_train,X_test,y_train,y_test = train_test_split(scaled_X,y,test_size=0.20,random_state= 12
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

Out[14]:

```
((80, 16), (80,), (21, 16), (21,))
```

In [15]:

```
X_train
```

Out[15]:

```
array([[-0.86103386, -0.49690399,  0.84372057, ..., -1.69841555,
        -0.38435306, -0.87859537],
       [ 1.16139451, -0.49690399, -1.18522652, ...,  0.58878406,
        -0.38435306,  1.13818037],
       [-0.86103386, -0.49690399,  0.84372057, ..., -1.69841555,
        -0.38435306, -0.87859537],
       ...,
       [ 1.16139451, -0.49690399, -1.18522652, ..., -1.69841555,
        -0.38435306,  1.13818037],
       [-0.86103386, -0.49690399,  0.84372057, ...,  0.58878406,
        -0.38435306, -0.87859537],
       [-0.86103386, -0.49690399,  0.84372057, ...,  0.58878406,
        -0.38435306,  1.13818037]])
```

In [16]:

```
X_test
```

Out[16]:

```
array([[-0.86103386, -0.49690399,  0.84372057, -0.82663978, -0.55829053,
         1.34370962,  0.89642146, -1.2349089 , -2.14734979, -1.95180015,
        -0.29329423, -0.44986771,  2.54951445, -1.69841555, -0.38435306,
         1.13818037],
       [-0.86103386, -0.49690399,  0.84372057, -0.82663978, -0.55829053,
         1.34370962,  0.89642146,  0.80977633,  0.46569032, -1.95180015,
        -0.29329423,  2.22287572, -1.40443503,  0.58878406, -0.38435306,
        -0.87859537],
       [ 1.16139451, -0.49690399, -1.18522652,  1.20971676, -0.55829053,
        -0.74420841, -1.1155467 ,  0.80977633,  0.46569032,  0.51234754,
        -0.29329423, -0.44986771, -0.41594766,  0.58878406, -0.38435306,
        -0.87859537],
       [ 1.16139451, -0.49690399, -1.18522652,  1.20971676, -0.55829053,
        -0.74420841, -1.1155467 ,  0.80977633,  0.46569032,  0.51234754,
        -0.29329423, -0.44986771,  0.57253971, -1.69841555,  2.60177454,
        -0.87859537],
       [ 1.16139451, -0.49690399, -1.18522652,  1.20971676,  1.79118211,
        -0.74420841, -1.1155467 ,  0.80977633,  0.46569032,  0.51234754,
        -0.29329423, -0.44986771, -0.41594766,  0.58878406, -0.38435306,
        -0.87859537],
       [-0.86103386,  2.01246118,  0.84372057, -0.82663978,  1.79118211,
         1.34370962,  0.89642146, -1.2349089 ,  0.46569032,  0.51234754,
        -0.29329423, -0.44986771, -0.41594766,  0.58878406, -0.38435306,
        -0.87859537],
       [ 1.16139451, -0.49690399, -1.18522652,  1.20971676, -0.55829053,
        -0.74420841,  0.89642146,  0.80977633,  0.46569032,  0.51234754,
        -0.29329423, -0.44986771,  0.57253971,  0.58878406, -0.38435306,
         1.13818037],
       [ 1.16139451, -0.49690399,  0.84372057, -0.82663978,  1.79118211,
        -0.74420841, -1.1155467 , -1.2349089 , -2.14734979,  0.51234754,
        -0.29329423, -0.44986771,  1.56102708, -1.69841555, -0.38435306,
        -0.87859537],
       [ 1.16139451, -0.49690399, -1.18522652,  1.20971676, -0.55829053,
        -0.74420841,  0.89642146,  0.80977633,  0.46569032,  0.51234754,
        -0.29329423, -0.44986771,  0.57253971,  0.58878406, -0.38435306,
         1.13818037],
       [-0.86103386, -0.49690399,  0.84372057, -0.82663978, -0.55829053,
         1.34370962,  0.89642146,  0.80977633,  0.46569032,  0.51234754,
         3.40954542, -0.44986771,  0.57253971, -1.69841555, -0.38435306,
        -0.87859537],
       [-0.86103386,  2.01246118,  0.84372057, -0.82663978,  1.79118211,
        -0.74420841, -1.1155467 , -1.2349089 ,  0.46569032,  0.51234754,
        -0.29329423, -0.44986771, -0.41594766,  0.58878406,  2.60177454,
        -0.87859537],
       [ 1.16139451, -0.49690399,  0.84372057, -0.82663978,  1.79118211,
        -0.74420841, -1.1155467 , -1.2349089 , -2.14734979,  0.51234754,
        -0.29329423, -0.44986771,  1.56102708, -1.69841555, -0.38435306,
        -0.87859537],
       [-0.86103386, -0.49690399,  0.84372057, -0.82663978, -0.55829053,
         1.34370962,  0.89642146, -1.2349089 , -2.14734979, -1.95180015,
         3.40954542, -0.44986771, -1.40443503, -1.69841555, -0.38435306,
        -0.87859537],
       [-0.86103386,  2.01246118,  0.84372057, -0.82663978,  1.79118211,
         1.34370962,  0.89642146, -1.2349089 ,  0.46569032,  0.51234754,
        -0.29329423, -0.44986771, -0.41594766,  0.58878406, -0.38435306,
```

```
        -0.87859537],
       [ 1.16139451, -0.49690399, -1.18522652,  1.20971676, -0.55829053,
        -0.74420841,  0.89642146,  0.80977633,  0.46569032,  0.51234754,
        -0.29329423, -0.44986771,  0.57253971, -1.69841555, -0.38435306,
         1.13818037],
       [-0.86103386,  2.01246118,  0.84372057, -0.82663978,  1.79118211,
        -0.74420841,  0.89642146, -1.2349089 ,  0.46569032,  0.51234754,
        -0.29329423, -0.44986771, -0.41594766,  0.58878406, -0.38435306,
        -0.87859537],
       [-0.86103386, -0.49690399,  0.84372057, -0.82663978, -0.55829053,
         1.34370962, -1.1155467 ,  0.80977633,  0.46569032, -1.95180015,
        -0.29329423,  2.22287572, -1.40443503,  0.58878406, -0.38435306,
        -0.87859537],
       [-0.86103386, -0.49690399,  0.84372057, -0.82663978, -0.55829053,
        -0.74420841,  0.89642146,  0.80977633,  0.46569032,  0.51234754,
        -0.29329423, -0.44986771, -1.40443503,  0.58878406, -0.38435306,
        -0.87859537],
       [ 1.16139451, -0.49690399, -1.18522652,  1.20971676, -0.55829053,
        -0.74420841, -1.1155467 ,  0.80977633,  0.46569032,  0.51234754,
        -0.29329423, -0.44986771,  0.57253971,  0.58878406, -0.38435306,
         1.13818037],
       [ 1.16139451, -0.49690399, -1.18522652,  1.20971676, -0.55829053,
        -0.74420841,  0.89642146,  0.80977633,  0.46569032,  0.51234754,
        -0.29329423, -0.44986771,  0.57253971,  0.58878406, -0.38435306,
        -0.87859537],
       [-0.86103386, -0.49690399,  0.84372057, -0.82663978, -0.55829053,
         1.34370962,  0.89642146,  0.80977633,  0.46569032, -1.95180015,
        -0.29329423,  2.22287572, -1.40443503,  0.58878406, -0.38435306,
        -0.87859537]])
```

In [17]:

```
knn_model = KNeighborsClassifier(n_neighbors=4)
knn_model.fit(X_train,y_train)
y_pred = knn_model.predict(X_test)
print("Accuracy score: ", round(accuracy_score(y_test,y_pred),4))
```

```
Accuracy score:  0.9524
```

Model Accuracy score without standardization is : 0.9524

Model Accuracy score with standardization is : 0.9524

In [ ]:

```

```