In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
import seaborn as sns
import numpy as np
```

In [22]:
```
zoo_data = pd.read_csv('Zoo.csv')
zoo_data
```

Out[22]:

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

101 rows × 18 columns

In [23]:
```
# initial analysis
zoo_data.shape
```

Out[23]:
```
(101, 18)
```

In [24]:
```
zoo_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101 entries, 0 to 100
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   animal name  101 non-null    object
 1   hair         101 non-null    int64 
 2   feathers     101 non-null    int64 
 3   eggs         101 non-null    int64 
 4   milk         101 non-null    int64 
 5   airborne     101 non-null    int64 
 6   aquatic      101 non-null    int64 
 7   predator     101 non-null    int64 
 8   toothed      101 non-null    int64 
 9   backbone     101 non-null    int64 
 10  breathes     101 non-null    int64 
 11  venomous     101 non-null    int64 
 12  fins         101 non-null    int64 
 13  legs         101 non-null    int64 
```

In [25]:
```
zoo_data.isna().sum()
```

Out[25]:
```
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 [26]:
```
zoo_data.dtypes
```

Out[26]:
```
animal name    object
hair            int64
feathers        int64
eggs            int64
milk            int64
airborne        int64
aquatic         int64
predator        int64
toothed         int64
backbone        int64
breathes        int64
venomous        int64
fins            int64
legs            int64
tail            int64
domestic        int64
catsize         int64
type            int64
dtype: object
```

In [27]:
```
zoo_data = zoo_data.rename({'animal name':'animal_name'})
zoo_data
```

Out[27]:

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

101 rows × 18 columns

In [29]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [31]:
```
zoo_data['animal name'] = le.fit_transform(zoo_data['animal name'])
zoo_data
```

Out[31]:

	animal name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes
0	0	1	0	0	1	0	0	1	1	1	1
1	1	1	0	0	1	0	0	0	1	1	1
2	2	0	0	1	0	0	1	1	1	1	0
3	3	1	0	0	1	0	0	1	1	1	1
4	4	1	0	0	1	0	0	1	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...
96	95	1	0	0	1	0	0	0	1	1	1
97	96	1	0	1	0	1	0	0	0	0	1
98	97	1	0	0	1	0	0	1	1	1	1
99	98	0	0	1	0	0	0	0	0	0	1
100	99	0	1	1	0	1	0	0	0	1	1

101 rows × 18 columns

In [32]:
```
zoo_data.dtypes
```

Out[32]:
```
animal name    int32
hair           int64
feathers       int64
eggs           int64
milk           int64
airborne       int64
aquatic        int64
predator       int64
toothed        int64
backbone       int64
breathes       int64
venomous       int64
fins           int64
legs           int64
tail           int64
domestic       int64
catsize        int64
type           int64
dtype: object
```

Model Building

In [39]:
```
X = zoo_data.drop(['type'], axis=1)
y = zoo_data['type']
```

In [40]:
```
X
```

Out[40]:

	animal name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes
0	0	1	0	0	1	0	0	1	1	1	1
1	1	1	0	0	1	0	0	0	1	1	1
2	2	0	0	1	0	0	1	1	1	1	0
3	3	1	0	0	1	0	0	1	1	1	1
4	4	1	0	0	1	0	0	1	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...
96	95	1	0	0	1	0	0	0	1	1	1
97	96	1	0	1	0	1	0	0	0	0	1
98	97	1	0	0	1	0	0	1	1	1	1
99	98	0	0	1	0	0	0	0	0	0	1
100	99	0	1	1	0	1	0	0	0	1	1

101 rows × 17 columns

In [41]:
```
y
```

Out[41]:
```
0      1
1      1
2      4
3      1
4      1
      ..
96     1
97     6
98     1
99     7
100    2
Name: type, Length: 101, dtype: int64
```

In [43]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=0.20, random_state
```

In [44]:
```
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

Out[44]:
```
((80, 17), (80,), (21, 17), (21,))
```

Model Training | Testing | Evaluation without NORMALIZATION

Generating a Model with K=3

In [54]:
```
knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train,y_train)
```

Out[54]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [55]:
```
y_test_pred = knn_model.predict(X_test)
print("Accuracy score : " , round(accuracy_score(y_test,y_test_pred),4))
```

```
Accuracy score :  0.5238
```

In [ ]:
```

```

Model Training | Testing | Evaluation with NORMALIZATION

Generating a Model with K=3

In [56]:
```
scaler = StandardScaler()
scaled_X = scaler.fit_transform(X)
```

In [57]:
```
X_train,X_test,y_train,y_test= train_test_split(scaled_X,y, test_size=0.20, randc
X_train.shape , y_train.shape, X_test.shape, y_test.shape
```

Out[57]:
```
((80, 17), (80,), (21, 17), (21,))
```

In [58]:
```
X_train
```

Out[58]:
```
array([[ 0.12983751,  1.16139451, -0.49690399, ...,  0.58878406,
        -0.38435306, -0.87859537],
       [-0.8415394 , -0.86103386, -0.49690399, ..., -1.69841555,
        -0.38435306, -0.87859537],
       [ 0.16452954,  1.16139451, -0.49690399, ...,  0.58878406,
        -0.38435306,  1.13818037],
       ...,
       [-1.50068801,  1.16139451, -0.49690399, ...,  0.58878406,
         2.60177454,  1.13818037],
       [-0.80684737,  1.16139451, -0.49690399, ...,  0.58878406,
        -0.38435306, -0.87859537],
       [ 0.85837019,  1.16139451, -0.49690399, ...,  0.58878406,
        -0.38435306,  1.13818037]])
```

In [59]:
```
X_test
```

Out[59]:
```
array([[-1.11907566,  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],
       [-0.32115891, -0.86103386,  2.01246118,  0.84372057, -0.82663978,
        -0.55829053, -0.74420841,  0.89642146, -1.2349089 ,  0.46569032,
         0.51234754, -0.29329423, -0.44986771, -0.41594766,  0.58878406,
        -0.38435306, -0.87859537],
       [ 1.48282677,  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],
       [-1.22315175, -0.86103386, -0.49690399,  0.84372057, -0.82663978,
        -0.55829053,  1.34370962,  0.89642146, -1.2349089 , -2.14734979,
        -1.95180015, -0.29329423, -0.44986771,  0.57253971, -1.69841555,
        -0.38435306, -0.87859537],
       [ 0.65021799,  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],
       [-0.66807924,  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,
         2.60177454,  1.13818037],
       [ 1.37875067, -0.86103386, -0.49690399,  0.84372057, -0.82663978,
        -0.55829053, -0.74420841, -1.1155467 , -1.2349089 ,  0.46569032,
         0.51234754, -0.29329423, -0.44986771,  0.57253971,  0.58878406,
        -0.38435306,  1.13818037],
       [-1.18845972, -0.86103386, -0.49690399,  0.84372057, -0.82663978,
        -0.55829053,  1.34370962,  0.89642146, -1.2349089 , -2.14734979,
        -1.95180015, -0.29329423, -0.44986771,  1.56102708, -1.69841555,
        -0.38435306, -0.87859537],
       [-0.98030753, -0.86103386,  2.01246118,  0.84372057, -0.82663978,
         1.79118211,  1.34370962, -1.1155467 , -1.2349089 ,  0.46569032,
         0.51234754, -0.29329423, -0.44986771, -0.41594766,  0.58878406,
        -0.38435306, -0.87859537],
       [ 0.37268173, -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],
       [-1.29253582, -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],
       [-1.43130395, -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],
       [-0.39054298,  1.16139451, -0.49690399,  0.84372057, -0.82663978,
         1.79118211, -0.74420841, -1.1155467 , -1.2349089 , -2.14734979,
         0.51234754,  3.40954542, -0.44986771,  1.56102708, -1.69841555,
         2.60177454, -0.87859537],
       [-1.39661192,  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],
        [-1.46599598, -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,
          2.60177454, -0.87859537],
        [ 0.71960206, -0.86103386,  2.01246118,  0.84372057, -0.82663978,
         -0.55829053, -0.74420841,  0.89642146, -1.2349089 ,  0.46569032,
          0.51234754, -0.29329423, -0.44986771, -0.41594766,  0.58878406,
         -0.38435306,  1.13818037],
        [ 1.55221083, -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,  1.13818037],
        [-0.87623143, -0.86103386, -0.49690399,  0.84372057, -0.82663978,
         -0.55829053, -0.74420841, -1.1155467 , -1.2349089 , -2.14734979,
          0.51234754, -0.29329423, -0.44986771,  1.56102708, -1.69841555,
         -0.38435306, -0.87859537],
        [ 1.27467457, -0.86103386,  2.01246118,  0.84372057, -0.82663978,
          1.79118211,  1.34370962, -1.1155467 , -1.2349089 ,  0.46569032,
          0.51234754, -0.29329423, -0.44986771, -0.41594766,  0.58878406,
         -0.38435306,  1.13818037],
        [-1.15376769, -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],
        [ 1.17059848,  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]])
```

In [60]:
```
knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train,y_train)
```

Out[60]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [61]:
```
y_test_pred = knn_model.predict(X_test)
print("Accuracy score : " , round(accuracy_score(y_test,y_test_pred),4))
```

```
Accuracy score :  1.0
```

How to pickup Optimum no. of K?

In [62]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [65]:
```
neighbors = list(range(1,30))
cv_scores = []

for i in neighbors:
    knn_model = KNeighborsClassifier(n_neighbors = i)
    cv_score = cross_val_score(estimator = knn_model , X = scaled_X , y=y, cv=5)
    cv_scores.append(cv_score.mean())
```

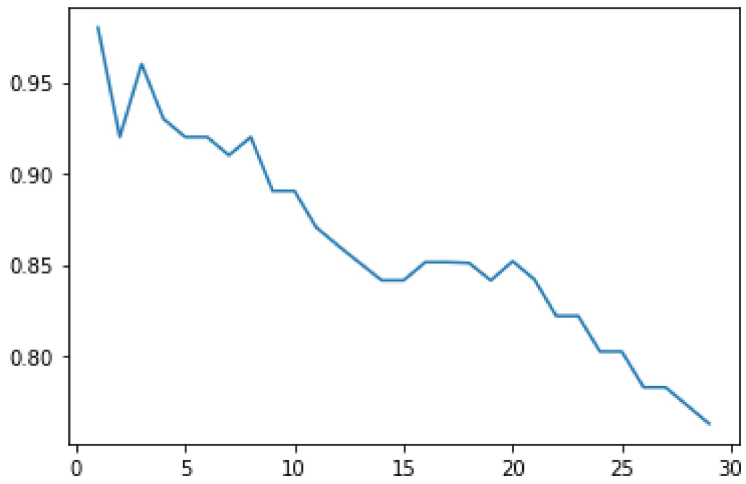
In [66]:
```
cv_scores
```

Out[66]:
```
[0.9800000000000001,
 0.9200000000000002,
 0.96,
 0.93,
 0.9200000000000002,
 0.9199999999999999,
 0.9099999999999999,
 0.9200000000000002,
 0.8904761904761905,
 0.8904761904761905,
 0.8704761904761906,
 0.8604761904761904,
 0.8509523809523809,
 0.8414285714285714,
 0.8414285714285714,
 0.8514285714285714,
 0.8514285714285714,
 0.8509523809523809,
 0.8414285714285714,
 0.8519047619047619,
 0.8419047619047617,
 0.8219047619047618,
 0.8219047619047618,
 0.8023809523809525,
 0.8023809523809525,
 0.7828571428571429,
 0.7828571428571429,
 0.7728571428571429,
 0.7628571428571428]
```

In [67]:
```
neighbors[cv_scores.index(max(cv_scores))]
```

Out[67]:
```
1
```

In [68]:
```
plt.plot(neighbors,cv_scores)
plt.show()
```



In [69]:
```
knn_model = KNeighborsClassifier(n_neighbors=1)
knn_model.fit(X_train,y_train)
y_test_pred = knn_model.predict(X_test)
print("Accuracy Score : " , round(accuracy_score(y_test,y_test_pred),4))
```

```
Accuracy Score :  0.9524
```

In [ ]:
```

```