In [3]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

import os
print(os.listdir("C:\\Users\\Admin\\Downloads\\Assignment 5"))
%matplotlib inline
```

```
['glass.csv', 'problem_statement(Glass).txt', 'Problem_statement(salary_data).txt', 'Pro
blem_Statement(Zoo).txt', 'SalaryData_Test.csv', 'SalaryData_Train.csv', 'Zoo.csv']
```

In [2]:

```
zoo=pd.read_csv("C:\\Users\\Admin\\Downloads\\Assignment 5\\Zoo.csv")
zoo.head(10)
```

Out[2]:

	animalname	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
5	buffalo	1	0	0	1	0	0	0	1	1	1
6	calf	1	0	0	1	0	0	0	1	1	1
7	carp	0	0	1	0	0	1	0	1	1	0
8	catfish	0	0	1	0	0	1	1	1	1	0
9	cavy	1	0	0	1	0	0	0	1	1	1

In [4]:

```
zoo.head()
```

Out[4]:

	animalname	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

In [5]:

```
zoo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101 entries, 0 to 100
Data columns (total 18 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   animalname  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 
 14  tail        101 non-null    int64 
 15  domestic    101 non-null    int64 
 16  catsize     101 non-null    int64 
 17  type        101 non-null    int64 
dtypes: int64(17), object(1)
memory usage: 14.3+ KB
```

In [6]:

```
zoo.describe()
```

Out[6]:

	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed
count	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000
mean	0.425743	0.198020	0.584158	0.405941	0.237624	0.356436	0.554455	0.603960
std	0.496921	0.400495	0.495325	0.493522	0.427750	0.481335	0.499505	0.491512
min	0.000000	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	0.000000
50%	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000
75%	1.000000	0.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [8]:

```
zoo.drop("animalname",axis=1,inplace=True)
```

In [9]:

```
color_list = [("red" if i ==1 else "blue" if i ==0 else "yellow" ) for i in zoo.hair]
```

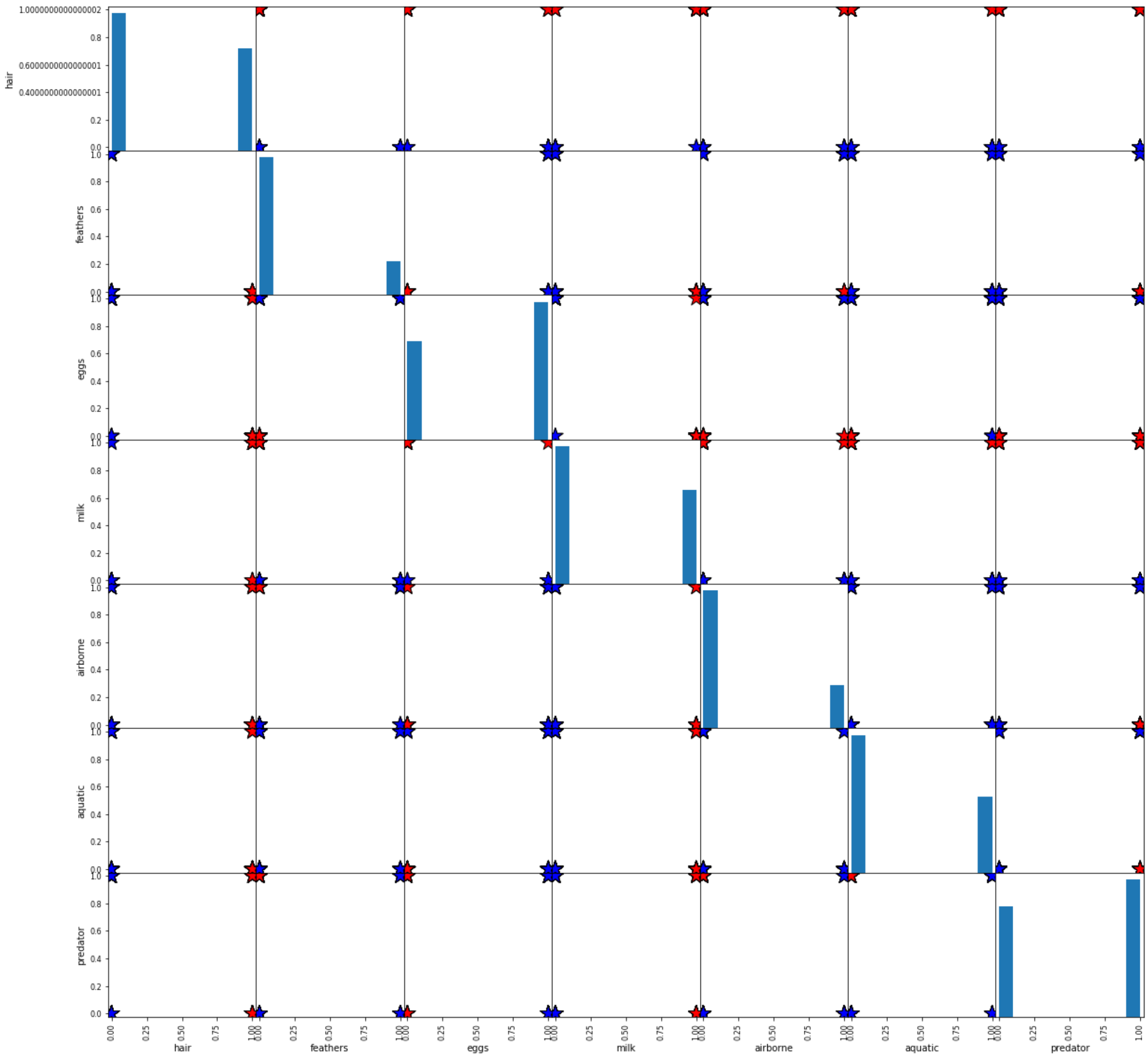
In [10]:

```
unique_list = list(set(color_list))
unique_list
```

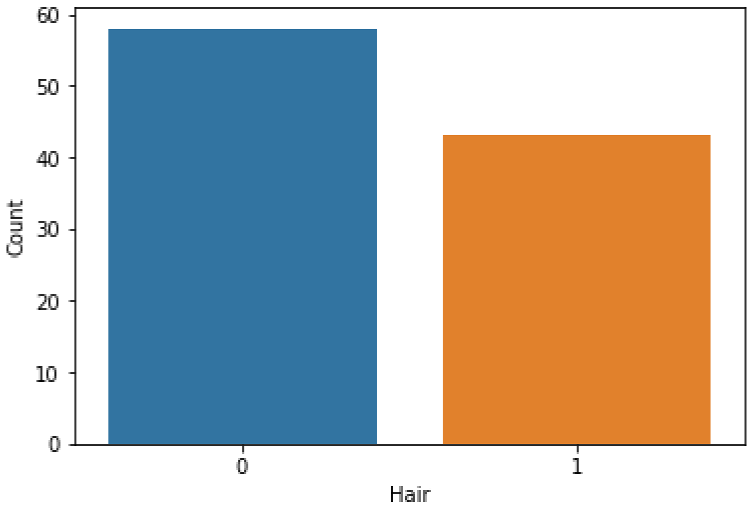
Out[10]:

```
['blue', 'red']
```

In [11]:
```
pd.plotting.scatter_matrix(zoo.iloc[:,:7],
                                       c=color_list,
                                       figsize= [20,20],
                                       diagonal='hist',
                                       alpha=1,
                                       s = 300,
                                       marker = '*',
                                       edgecolor= "black")
plt.show()
```



In [12]:
```
sns.countplot(x="hair", data=zoo)
plt.xlabel("Hair")
plt.ylabel("Count")
plt.show()
zoo.loc[:,'hair'].value_counts()
```



Out[12]:
```
0    58
1    43
Name: hair, dtype: int64
```

In [13]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 1)
x,y = zoo.loc[:,zoo.columns != 'hair'], zoo.loc[:,'hair']
knn.fit(x,y)
prediction = knn.predict(x)
print("Prediction = ",prediction)
```

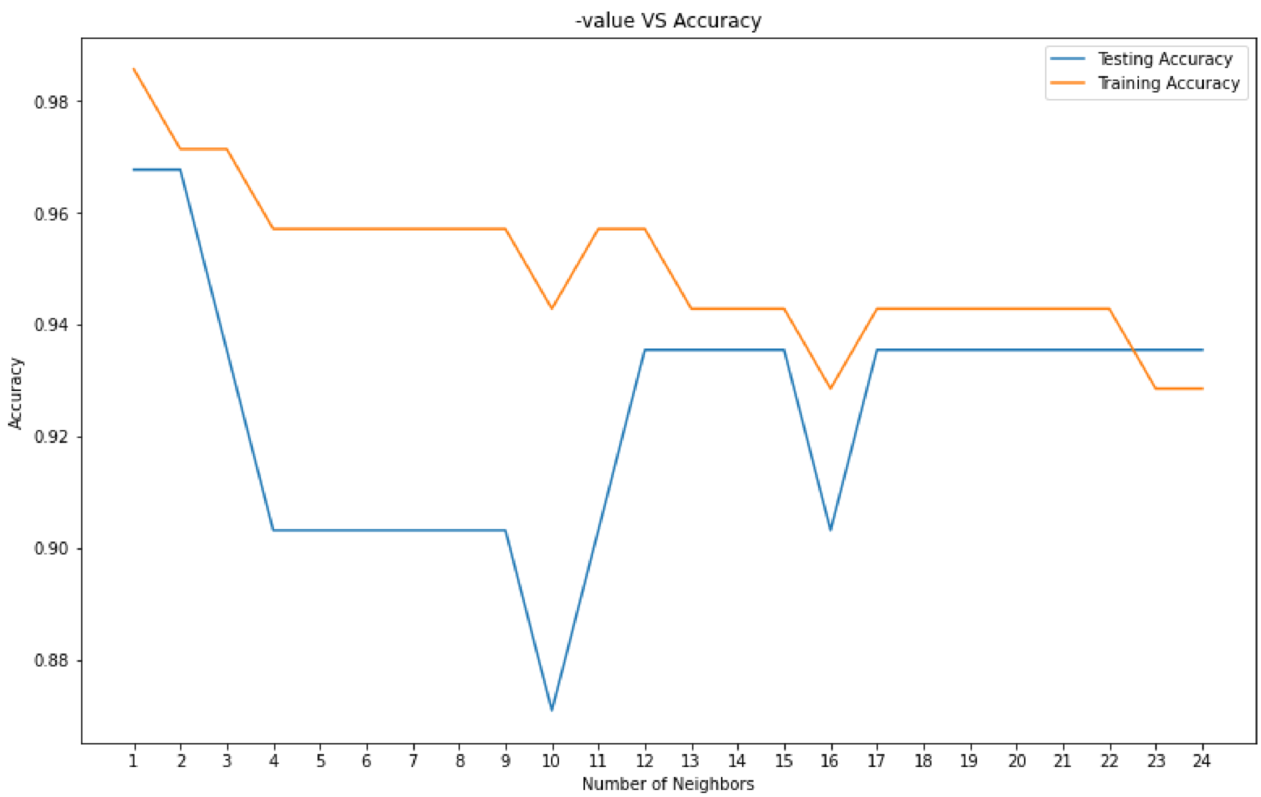
```
Prediction =  [1 1 0 1 1 1 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 1 0 1 1 0 0 1 1
 0 0 1 0 0 0 0 1 1 0 1 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 0 0
 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 0]
```

In [14]:
```
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 = 1)
knn = KNeighborsClassifier(n_neighbors = 1)
x,y = zoo.loc[:,zoo.columns != 'hair'], zoo.loc[:,'hair']
knn.fit(x_train,y_train)
prediction = knn.predict(x_test)
print('With KNN (K=1) accuracy is: ',knn.score(x_test,y_test)) # accuracy
```

```
With KNN (K=1) accuracy is:  0.967741935483871
```

In [15]:
```
k_values = np.arange(1,25)
train_accuracy = []
test_accuracy = []

for i, k in enumerate(k_values):
    # k from 1 to 25(exclude)
    knn = KNeighborsClassifier(n_neighbors=k)
    # Fit with knn
    knn.fit(x_train,y_train)
    #train accuracy
    train_accuracy.append(knn.score(x_train, y_train))
    # test accuracy
    test_accuracy.append(knn.score(x_test, y_test))

    # Plot
plt.figure(figsize=[13,8])
plt.plot(k_values, test_accuracy, label = 'Testing Accuracy')
```

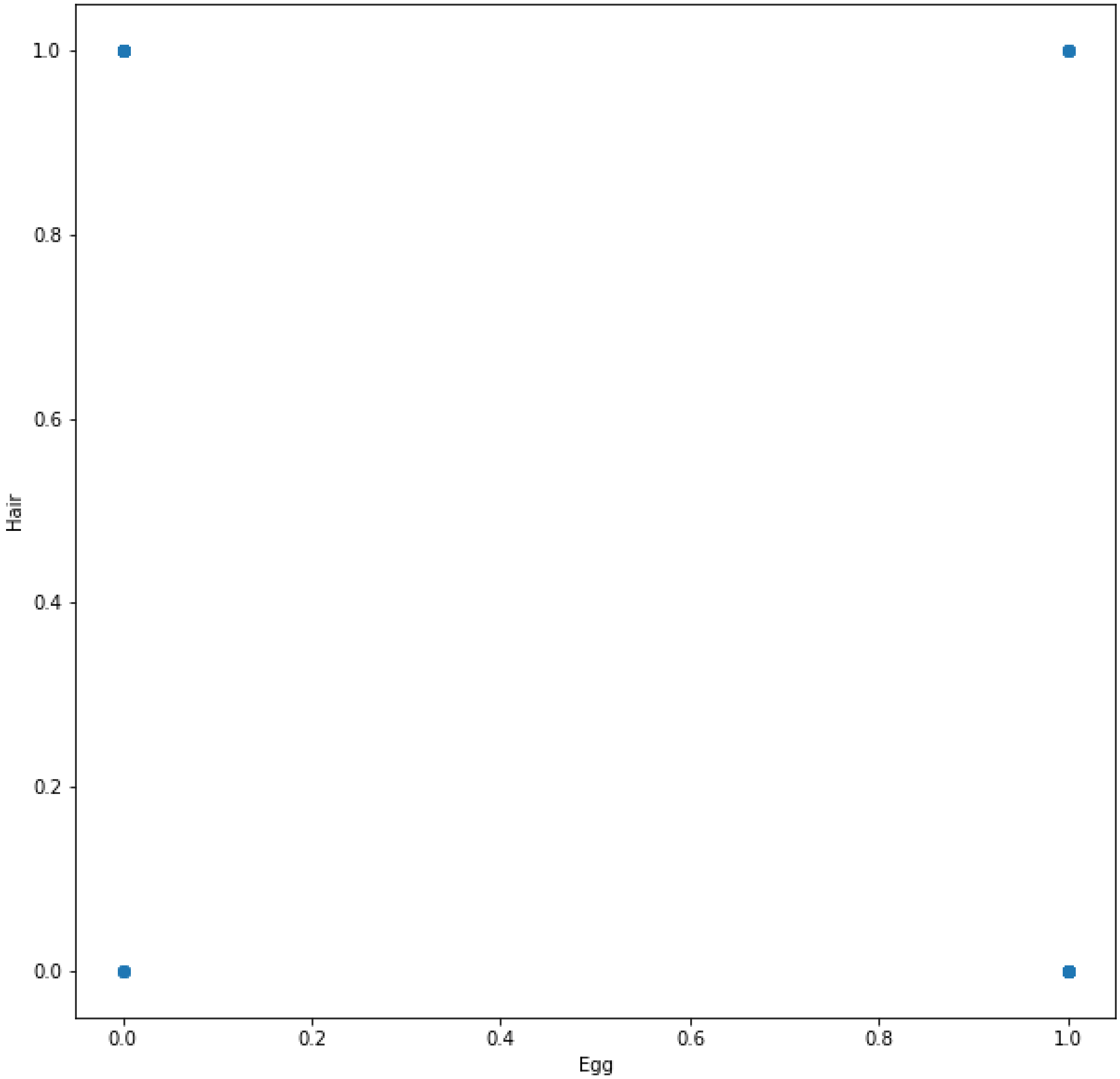
```
plt.plot(k_values, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.title('-value VS Accuracy')
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.savefig('graph.png')
plt.show()
print("Best accuracy is {} with K = {}".format(np.max(test_accuracy),1+test_accuracy.in
```




```
Best accuracy is 0.967741935483871 with K = 1
```

In [16]:

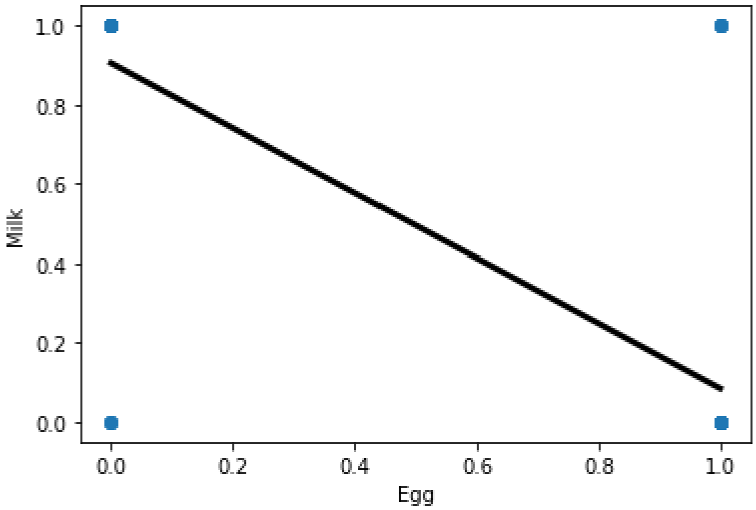
```
x = np.array(zoo.loc[:,"eggs"]).reshape(-1,1)
y = np.array(zoo.loc[:,'hair']).reshape(-1,1)

plt.figure(figsize=[10,10])
plt.scatter(x=x,y=y)
plt.xlabel('Egg')
plt.ylabel('Hair')
plt.show()
```



In [17]:

```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()

predict_space = np.linspace(min(x),max(x)).reshape(-1,1)
regression.fit(x,y)
predicted = regression.predict(predict_space)

print("R^2 Score: ",regression.score(x,y))

plt.plot(predict_space, predicted, color='black', linewidth=3)
plt.scatter(x=x,y=y)
plt.xlabel('Egg')
plt.ylabel('Milk')
plt.show()
```

```
R^2 Score:  0.6681125904754137
```



In [18]:

```
from sklearn.model_selection import cross_val_score
regression = LinearRegression()
k=5
cv_result = cross_val_score(regression,x,y,cv=k)
print("CV Scores: ",cv_result)
print("CV Average: ",np.sum(cv_result)/k)
```

```
CV Scores:  [0.80171562 0.61914032 0.79243817 0.24939434 0.76176534]
CV Average:  0.6448907578047475
```

In [19]:

```
from sklearn.linear_model import Ridge
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state = 2, test_size = 0.3)
ridge = Ridge(alpha= 0.001,normalize = True)
ridge.fit(x_train,y_train)
ridge_predict = ridge.predict(x_test)
print("Ridge Score: ",ridge.score(x_test,y_test))
```

```
Ridge Score:  0.930239727992853
```

In [21]:

```
from sklearn.linear_model import Lasso
x = np.array(zoo.loc[:,['eggs','airborne','fins','legs',"hair","type"]])
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state = 3, test_size = 0.3)
lasso = Lasso(alpha = 0.0001, normalize = True)
lasso.fit(x_train,y_train)
ridge_predict = lasso.predict(x_test)
print('Lasso score: ',lasso.score(x_test,y_test))
print('Lasso coefficients: ',lasso.coef_)
```

```
Lasso score:  0.9999970989932222
Lasso coefficients:  [-0.         -0.         -0.          0.          0.99830154 -0.
]
```

In [22]:

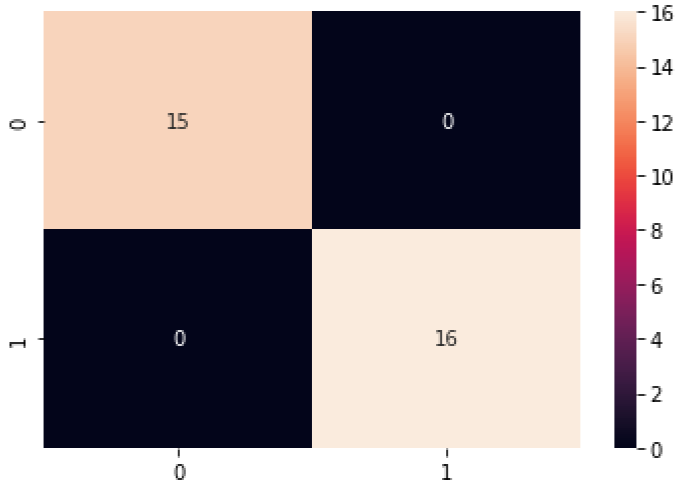
```
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.ensemble import RandomForestClassifier
x,y = zoo.loc[:,zoo.columns != "hair"], zoo.loc[:,"hair"]
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3,random_state = 1 )
rf = RandomForestClassifier(random_state = 4)
rf.fit(x_train,y_train)
y_pred = rf.predict(x_test)
cm = confusion_matrix(y_test,y_pred)
```

```
print("Confisuon Matrix: \n",cm)
print("Classification Report: \n",classification_report(y_test,y_pred))
```

```
Confisuon Matrix:
 [[15  0]
 [ 0 16]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        16

    accuracy                           1.00        31
   macro avg       1.00      1.00      1.00        31
weighted avg       1.00      1.00      1.00        31
```

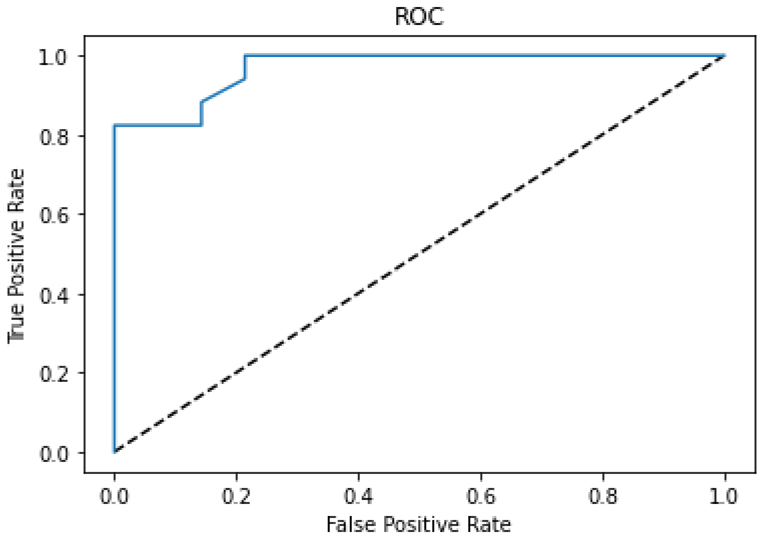
In [23]:

```
sns.heatmap(cm,annot=True,fmt="d")
plt.show()
```



In [24]:

```
from sklearn.metrics import roc_curve
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
#hair = 1 no = 0
x,y = zoo.loc[:,(zoo.columns != 'hair')], zoo.loc[:,'hair']
x_train,x_test,y_train,y_test = train_test_split(x, y, test_size = 0.3, random_state=42
logreg = LogisticRegression()
logreg.fit(x_train,y_train)
y_pred_prob = logreg.predict_proba(x_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
# Plot ROC curve
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.show()
```



In [25]:

```
from sklearn.model_selection import GridSearchCV
grid = {'n_neighbors': np.arange(1,50)}
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn, grid, cv=3) # GridSearchCV
knn_cv.fit(x,y)# Fit

# Print hyperparameter
print("Tuned hyperparameter k: {}".format(knn_cv.best_params_))
print("Best score: {}".format(knn_cv.best_score_))
```

```
Tuned hyperparameter k: {'n_neighbors': 1}
Best score: 0.9402852049910874
```

In [26]:

```
param_grid = {'C': np.logspace(-3, 3, 7), 'penalty': ['l1', 'l2']}
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.3,random_state =
logreg = LogisticRegression()
logreg_cv = GridSearchCV(logreg,param_grid,cv=3)
logreg_cv.fit(x_train,y_train)

# Print the optimal parameters and best score
print("Tuned hyperparameters : {}".format(logreg_cv.best_params_))
print("Best Accuracy: {}".format(logreg_cv.best_score_))
```

```
Tuned hyperparameters : {'C': 0.01, 'penalty': 'l2'}
Best Accuracy: 0.9299516908212562
```

In [27]:

```
df = pd.get_dummies(zoo)
df.head(10)
```

Out[27]:

	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fi
0	1	0	0	1	0	0	1	1	1	1	0	
1	1	0	0	1	0	0	0	1	1	1	0	
2	0	0	1	0	0	1	1	1	1	0	0	
3	1	0	0	1	0	0	1	1	1	1	0	
4	1	0	0	1	0	0	1	1	1	1	0	

	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fi
5	1	0	0	1	0	0	0	1	1	1	0	
6	1	0	0	1	0	0	0	1	1	1	0	
7	0	0	1	0	0	1	0	1	1	0	0	
8	0	0	1	0	0	1	1	1	1	0	0	
9	1	0	0	1	0	0	0	1	1	1	0	

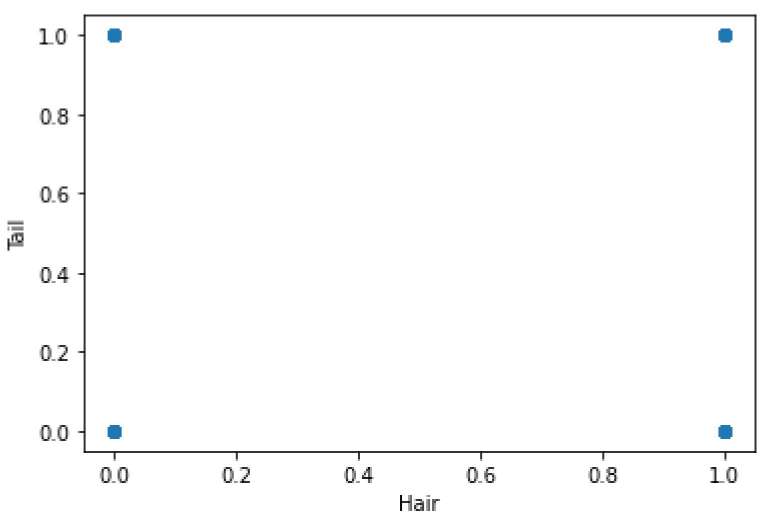
In [28]:

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
steps = [('scalar', StandardScaler()),
         ('SVM', SVC())]
pipeline = Pipeline(steps)
parameters = {'SVM__C':[1, 10, 100],
              'SVM__gamma':[0.1, 0.01]}
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state = 1)
cv = GridSearchCV(pipeline,param_grid=parameters,cv=3)
cv.fit(x_train,y_train)

y_pred = cv.predict(x_test)

print("Accuracy: {}".format(cv.score(x_test, y_test)))
print("Tuned Model Parameters: {}".format(cv.best_params_))
```

```
Accuracy: 0.9523809523809523
Tuned Model Parameters: {'SVM__C': 1, 'SVM__gamma': 0.01}
```

In [29]:

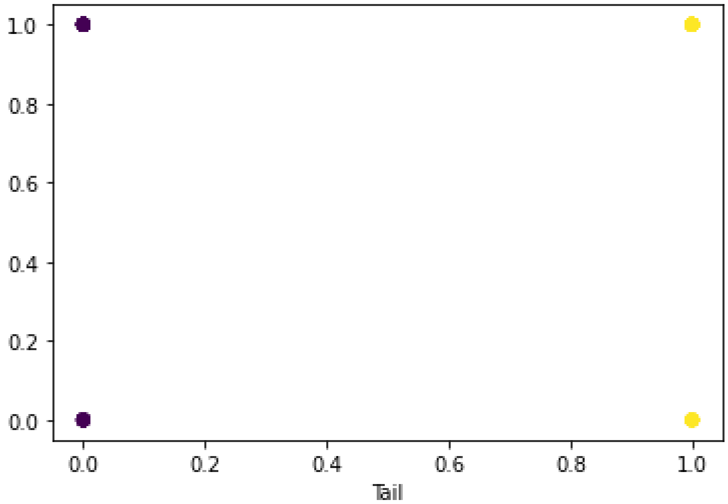
```
plt.scatter(zoo['hair'],zoo['tail'])
plt.xlabel('Hair')
plt.ylabel('Tail')
plt.show()
```



In [30]:

```
data2 = zoo.loc[:,['tail','hair']]
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 2)
```

```
kmeans.fit(data2)
labels = kmeans.predict(data2)
plt.scatter(zoo['hair'],zoo['tail'],c = labels)
plt.xlabel('Hair')
plt.xlabel('Tail')
plt.show()
```



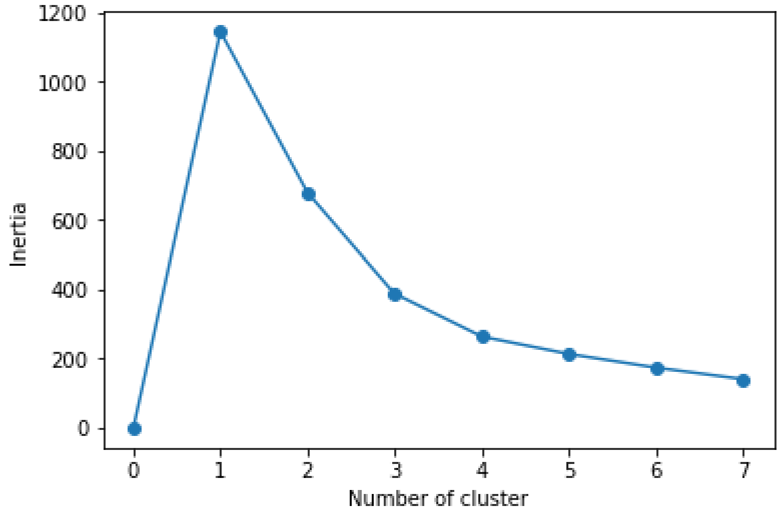
In [31]:

```
df = pd.DataFrame({'labels':labels,"hair":zoo['hair']})
ct = pd.crosstab(df['labels'],df['hair'])
print(ct)
```

```
hair     0   1
labels
0       58   0
1        0  43
```

In [32]:

```
inertia_list = np.empty(8)
for i in range(1,8):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(zoo)
    inertia_list[i] = kmeans.inertia_
plt.plot(range(0,8),inertia_list,'-o')
plt.xlabel('Number of cluster')
plt.ylabel('Inertia')
plt.show()
```



In [33]:

```
data2 = zoo.drop("hair",axis=1)
```

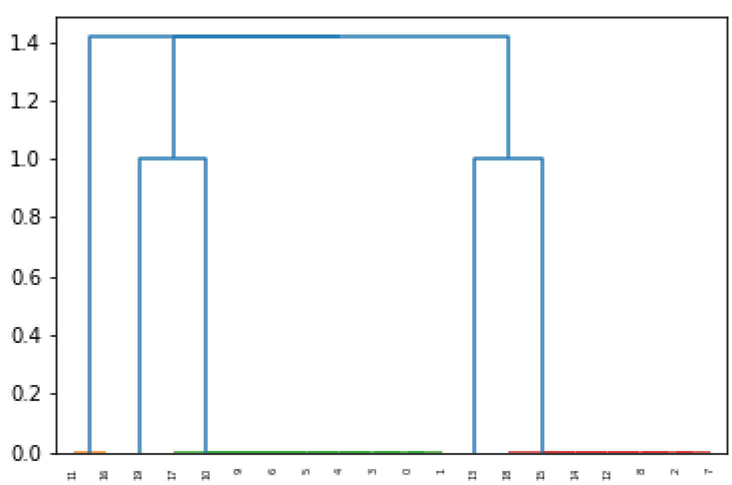
In [34]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
scalar = StandardScaler()
kmeans = KMeans(n_clusters = 2)
pipe = make_pipeline(scalar,kmeans)
pipe.fit(data2)
labels = pipe.predict(data2)
df = pd.DataFrame({'labels':labels,"hair":zoo['hair']})
ct = pd.crosstab(df['labels'],df['hair'])
print(ct)
```

```
hair     0   1
labels
0       56   4
1        2  39
```

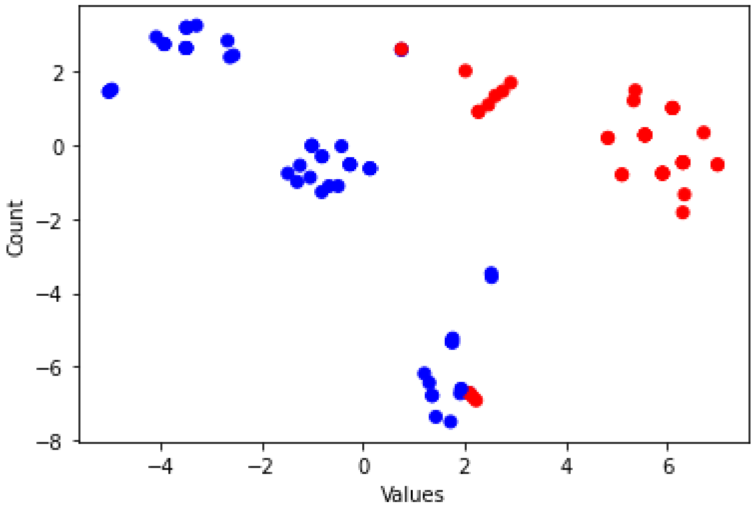
In [35]:

```
from scipy.cluster.hierarchy import linkage,dendrogram

merg = linkage(data2.iloc[:20,0:5],method = 'single')
dendrogram(merg, leaf_rotation = 90, leaf_font_size = 5)
plt.show()
```



In [36]:

```
from sklearn.manifold import TSNE
model = TSNE(learning_rate=100,random_state=42)
transformed = model.fit_transform(data2)
x = transformed[:,0]
y = transformed[:,1]
plt.scatter(x,y,c = color_list )
plt.xlabel('Values')
plt.ylabel('Count')
plt.show()
```



In [37]:

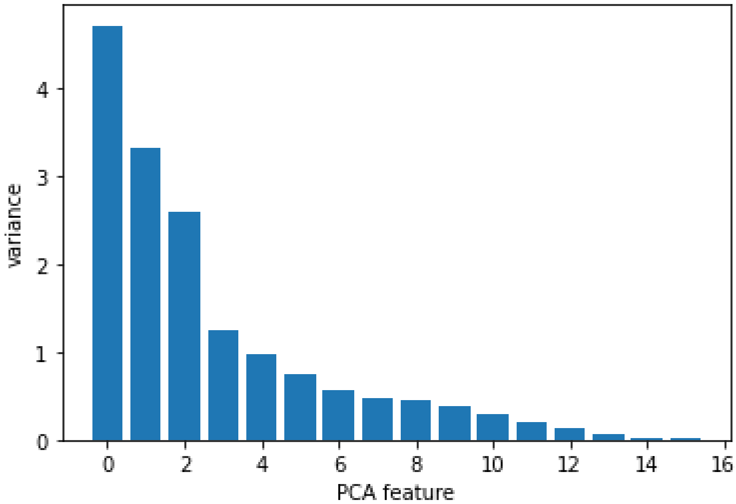
```
from sklearn.decomposition import PCA
model = PCA()
model.fit(data2[0:4])
transformed = model.transform(data2[0:4])
print('Principle components: ',model.components_)
```

```
Principle components:  [[-1.11022302e-16  1.77997984e-01 -1.77997984e-01  0.00000000e+00
   1.77997984e-01  5.75617345e-02  0.00000000e+00  0.00000000e+00
  -1.77997984e-01  0.00000000e+00  1.77997984e-01 -7.11991938e-01
   1.20436250e-01  0.00000000e+00 -1.77997984e-01  5.33993953e-01]
 [-3.33066907e-16 -7.92144437e-03  7.92144437e-03  0.00000000e+00
  -7.92144437e-03 -7.10368323e-01  0.00000000e+00  0.00000000e+00
   7.92144437e-03  0.00000000e+00 -7.92144437e-03  3.16857775e-02
   7.02446879e-01  0.00000000e+00  7.92144437e-03 -2.37643331e-02]
 [ 9.83538848e-01  5.50499658e-02 -4.07082498e-03 -0.00000000e+00
   4.07082498e-03  1.06099015e-01 -0.00000000e+00 -0.00000000e+00
  -4.07082498e-03 -0.00000000e+00  4.07082498e-03 -1.62832999e-02
   1.06099015e-01 -0.00000000e+00 -4.07082498e-03 -8.22121016e-02]
 [ 8.90295760e-02 -9.49149979e-01 -4.23156435e-02 -0.00000000e+00
   4.23156435e-02 -1.55559143e-01 -0.00000000e+00 -0.00000000e+00
  -4.23156435e-02 -0.00000000e+00  4.23156435e-02 -1.69262574e-01
  -1.55559143e-01 -0.00000000e+00 -4.23156435e-02  7.20268693e-02]]
```

In [38]:

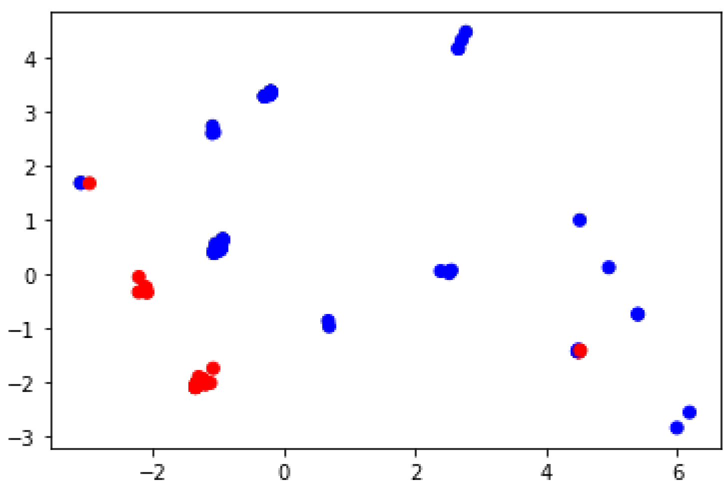
```
scaler = StandardScaler()
pca = PCA()
pipeline = make_pipeline(scaler,pca)
pipeline.fit(data2)

plt.bar(range(pca.n_components_), pca.explained_variance_)
plt.xlabel('PCA feature')
```

```python
plt.ylabel('variance')
plt.show()
```



In [39]:

```python
pca = PCA(n_components = 2)
pca.fit(data2)
transformed = pca.transform(data2)
x = transformed[:,0]
y = transformed[:,1]
plt.scatter(x,y,c = color_list)
plt.show()
```



In [ ]: