Assignment 13 - KNN

Prepare a model for glass classification using KNN Data

Description:

RI : refractive index

Na: Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 4-10)

Mg: Magnesium

Al: Aluminum

Si: Silicon

K:Potassium

Ca: Calcium

Ba: Barium

Fe: Iron

Type: Type of glass: (class attribute)

1 -- building_windows_float_processed

2 --building_windows_non_float_processed

3 --vehicle_windows_float_processed

4 --vehicle_windows_non_float_processed (none in this database)

5 --containers

6 --tableware

7 --headlamps

In [1]:

```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import seaborn as sns
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import cross_val_score
```

In [2]:

```
#load data
df = pd.read_csv('C:/Users/LENOVO/Documents/Custom Office Templates/glass.csv')
df.head()
```

Out[2]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

In [3]:

```
df.tail()
```

Out[3]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
209	1.51623	14.14	0.0	2.88	72.61	0.08	9.18	1.06	0.0	7
210	1.51685	14.92	0.0	1.99	73.06	0.00	8.40	1.59	0.0	7
211	1.52065	14.36	0.0	2.02	73.42	0.00	8.44	1.64	0.0	7
212	1.51651	14.38	0.0	1.94	73.61	0.00	8.48	1.57	0.0	7
213	1.51711	14.23	0.0	2.08	73.36	0.00	8.62	1.67	0.0	7

In [4]:

```
# value count for glass types
df.Type.value_counts()
```

Out[4]:

```
2    76
1    70
7    29
3    17
5    13
6     9
Name: Type, dtype: int64
```

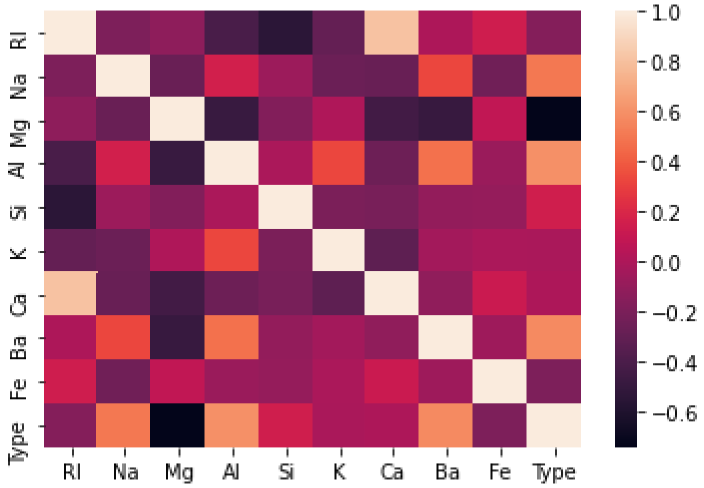
Data exploration and visualizaion

In [5]:

```
# correlation matrix
cor = df.corr()
sns.heatmap(cor)
```

Out[5]:

<AxesSubplot:>



We can notice that Ca and K values don't affect Type that much.

Also Ca and RI are highly correlated, this means using only RI is enough.

So we can go ahead and drop Ca, and also K.(performed later)

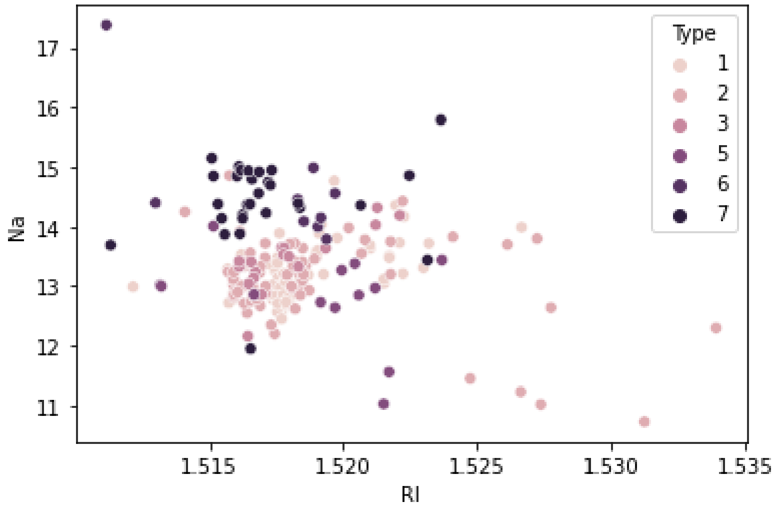
In [6]:

```
# Scatter plot of two features, and pairwise plot
sns.scatterplot(df['RI'],df['Na'],hue=df['Type'])
```

```
C:\Users\LENOVO\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Futur
eWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other a
rguments without an explicit keyword will result in an error or misinterpret
ation.
  warnings.warn(
```

Out[6]:

```
<AxesSubplot:xlabel='RI', ylabel='Na'>
```



Suppose we consider only RI, and Na values for classification for glass type.

From the above plot, We first calculate the nearest neighbors from the new data point to be calculated.

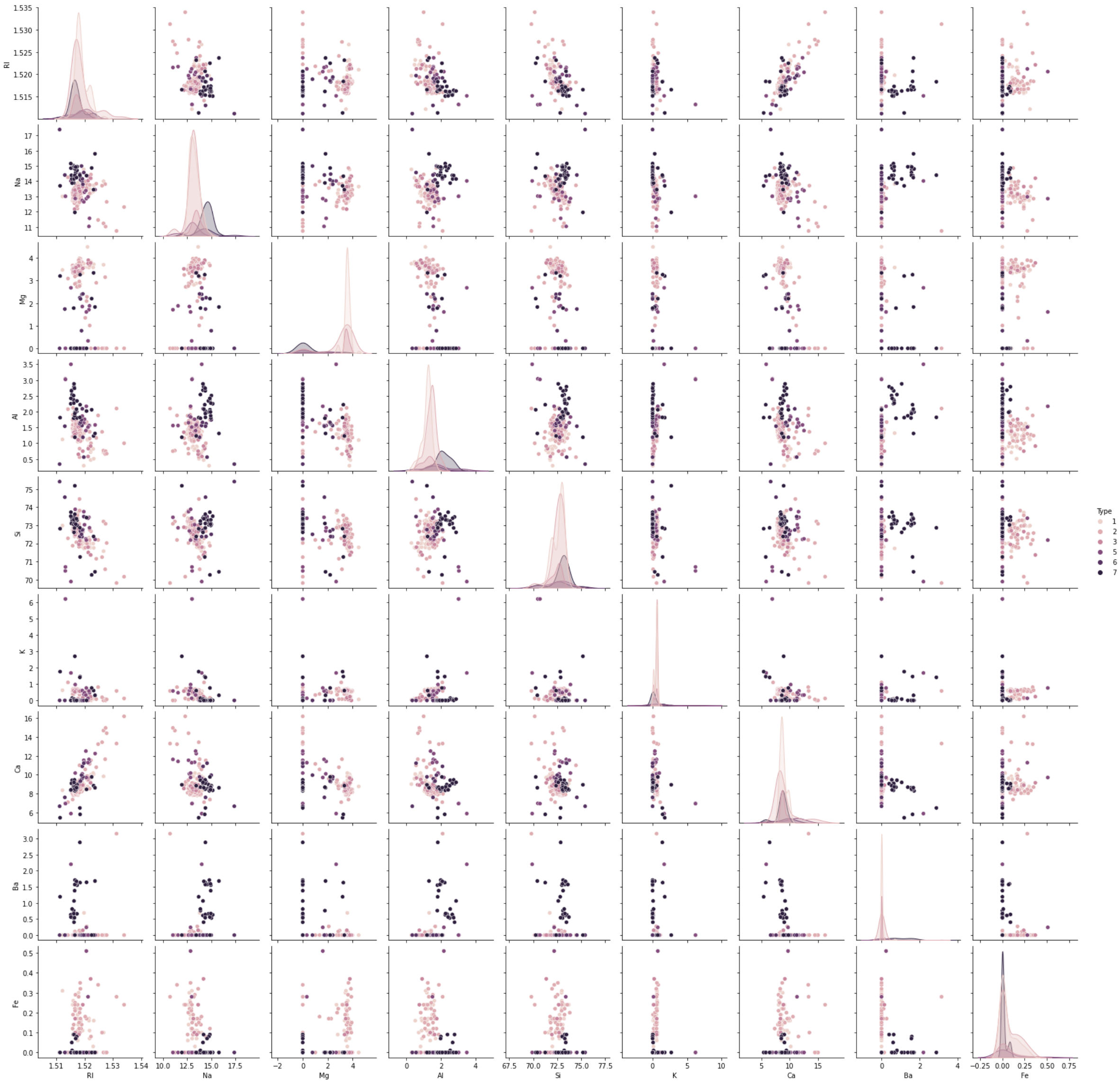
If the majority of nearest neighbors belong to a particular class, say type 4, then we classify the data point as type 4.

But there are a lot more than two features based on which we can classify. So let us take a look at pairwise plot to capture all the features.

In [7]:

```python
#pairwise plot of all the features
sns.pairplot(df,hue='Type')
plt.show()
```

```
C:\Users\LENOVO\anaconda3\lib\site-packages\seaborn\distributions.py:306: Us
erWarning: Dataset has 0 variance; skipping density estimate.
  warnings.warn(msg, UserWarning)
C:\Users\LENOVO\anaconda3\lib\site-packages\seaborn\distributions.py:306: Us
erWarning: Dataset has 0 variance; skipping density estimate.
  warnings.warn(msg, UserWarning)
C:\Users\LENOVO\anaconda3\lib\site-packages\seaborn\distributions.py:306: Us
erWarning: Dataset has 0 variance; skipping density estimate.
  warnings.warn(msg, UserWarning)
```



The pairplot shows that the data is not linear and KNN can be applied to get nearest neighbors and classify the glass types

Feature Scaling

Scaling is necessary for distance-based algorithms such as KNN. This is to avoid higher weightage being assigned to data with a higher magnitude.

Using standard scaler we can scale down to unit variance.

In [8]:

```
scaler = StandardScaler()
```

In [9]:

```
scaler.fit(df.drop('Type',axis=1))
```

Out[9]:

```
StandardScaler()
```

In [10]:

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

Out[10]:

```
StandardScaler()
```

In [11]:

```
#perform transformation
scaled_features = scaler.transform(df.drop('Type',axis=1))
scaled_features
```

Out[11]:

```
array([[ 0.87286765,  0.28495326,  1.25463857, ..., -0.14576634,
        -0.35287683, -0.5864509 ],
       [-0.24933347,  0.59181718,  0.63616803, ..., -0.79373376,
        -0.35287683, -0.5864509 ],
       [-0.72131806,  0.14993314,  0.60142249, ..., -0.82894938,
        -0.35287683, -0.5864509 ],
       ...,
       [ 0.75404635,  1.16872135, -1.86551055, ..., -0.36410319,
         2.95320036, -0.5864509 ],
       [-0.61239854,  1.19327046, -1.86551055, ..., -0.33593069,
         2.81208731, -0.5864509 ],
       [-0.41436305,  1.00915211, -1.86551055, ..., -0.23732695,
         3.01367739, -0.5864509 ]])
```

In [12]:

```
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
```

Out[12]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	
0	0.872868	0.284953	1.254639	-0.692442	-1.127082	-0.671705	-0.145766	-0.352877	-0.5864
1	-0.249333	0.591817	0.636168	-0.170460	0.102319	-0.026213	-0.793734	-0.352877	-0.5864
2	-0.721318	0.149933	0.601422	0.190912	0.438787	-0.164533	-0.828949	-0.352877	-0.5864
3	-0.232831	-0.242853	0.698710	-0.310994	-0.052974	0.112107	-0.519052	-0.352877	-0.5864
4	-0.312045	-0.169205	0.650066	-0.411375	0.555256	0.081369	-0.624699	-0.352877	-0.5864

Applying KNN

Drop features that are not required

Use random state while splitting the data to ensure reproducibility and consistency

Experiment with distance metrics - Euclidean, manhattan

In [13]:

```
dff = df_feat.drop(['Ca','K'],axis=1) #Removing features - Ca and K
X_train,X_test,y_train,y_test  = train_test_split(dff,df['Type'],test_size=0.3,random_state
#setting random state ensures split is same eveytime, so that the results are comparable
```

In [14]:

```
knn = KNeighborsClassifier(n_neighbors=4,metric='manhattan')
```

In [15]:

```
knn.fit(X_train,y_train)
```

Out[15]:

```
KNeighborsClassifier(metric='manhattan', n_neighbors=4)
```

In [16]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='manhattan',
                     metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                     weights='uniform')
```

Out[16]:

```
KNeighborsClassifier(metric='manhattan', n_neighbors=4)
```

In [17]:

```
y_pred = knn.predict(X_test)
```

In [18]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           1       0.69      0.90      0.78        20
           2       0.85      0.65      0.74        26
           3       0.00      0.00      0.00         3
           5       0.25      1.00      0.40         1
           6       0.50      0.50      0.50         2
           7       1.00      0.85      0.92        13

    accuracy                           0.74        65
   macro avg       0.55      0.65      0.56        65
weighted avg       0.77      0.74      0.74        65
```

In [19]:

```
accuracy_score(y_test,y_pred)
```

Out[19]:

```
0.7384615384615385
```

With this setup, We found the accuracy to be 73.84%

Finding the best K value

We can do this either -

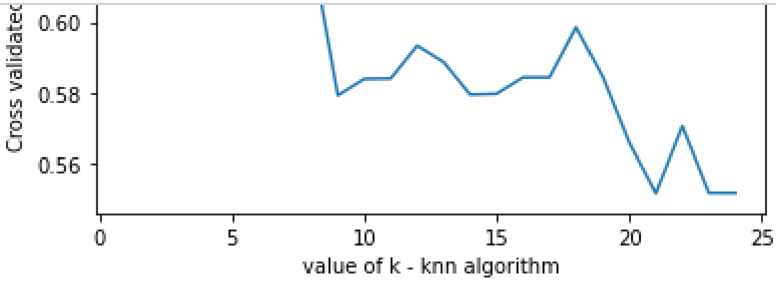
by plotting Accuracy

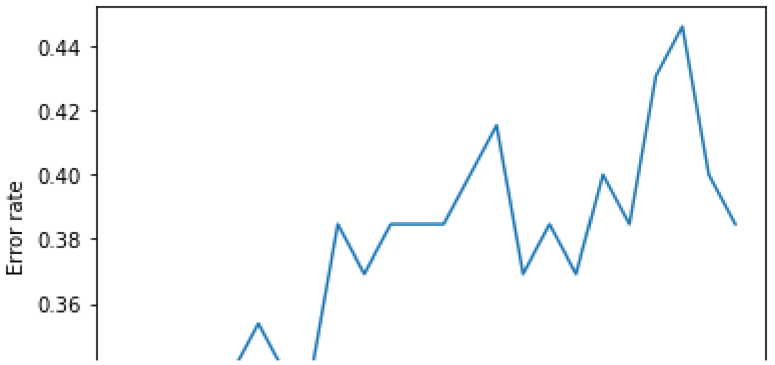
or by plotting the error rate

Note that plotting both is not required, however both are plottted here to show as an example.

In [20]:

```
k_range = range(1,25)
k_scores = []
error_rate =[]
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    #kscores - accuracy
    scores = cross_val_score(knn,dff,df['Type'],cv=5,scoring='accuracy')
    k_scores.append(scores.mean())

    #error rate
    knn.fit(X_train,y_train)
    y_pred = knn.predict(X_test)
    error_rate.append(np.mean(y_pred!=y_test))

#plot k vs accuracy
plt.plot(k_range,k_scores)
plt.xlabel('value of k - knn algorithm')
plt.ylabel('Cross validated accuracy score')
plt.show()

#plot k vs error rate
plt.plot(k_range,error_rate)
plt.xlabel('value of k - knn algorithm')
plt.ylabel('Error rate')
plt.show()
```





we can see that k=4 produces the most accurate results

Findings

Manhattan distance produced better results (improved accuracy - more than 5%)

Applying feature scaling improved accuracy by almost 5%.

The best k value was found to be 4.

Dropping ‘Ca’ produced better results by a bit, ‘K’ feature did not affect results in any way.

Also, we noticed that RI and Ca are highly correlated, this makes sense as it was found that the Refractive index of glass was found to increase with the increase in Cao

In [ ]:

```

```

Problem 2

Implement a KNN model to classify the animals in to categorie

In [21]:

```
#load data
zoo = pd.read_csv('C:/Users/LENOVO/Documents/Custom Office Templates/Zoo.csv')
zoo.head()
```

Out[21]:

	animal name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breath
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	

In [22]:

```
zoo.tail()
```

Out[22]:

	animal name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breat
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	

In [23]:

```
# value count for glass types
zoo.type.value_counts()
```

Out[23]:

```
1    41
2    20
4    13
7    10
6     8
3     5
5     4
Name: type, dtype: int64
```

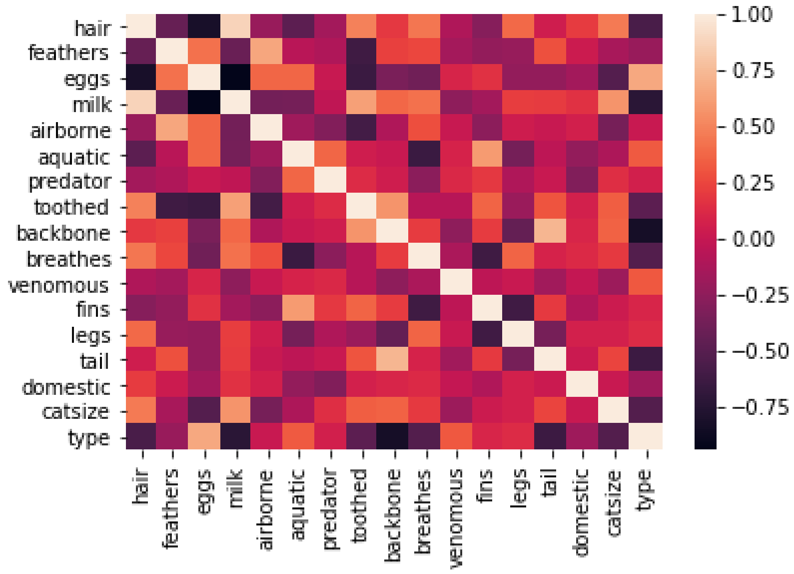
Data exploration and visualizaion

In [24]:

```
# correlation matrix
cor = zoo.corr()
sns.heatmap(cor)
```

Out[24]:

```
<AxesSubplot:>
```



In [25]:

```
zoo.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 
 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 [26]:

```
zoo.describe()
```

Out[26]:

	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 [27]:

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

In [28]:

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

In [29]:

```
# With this set function we find unique values in a list...
unique_list = list(set(color_list))
unique_list
```

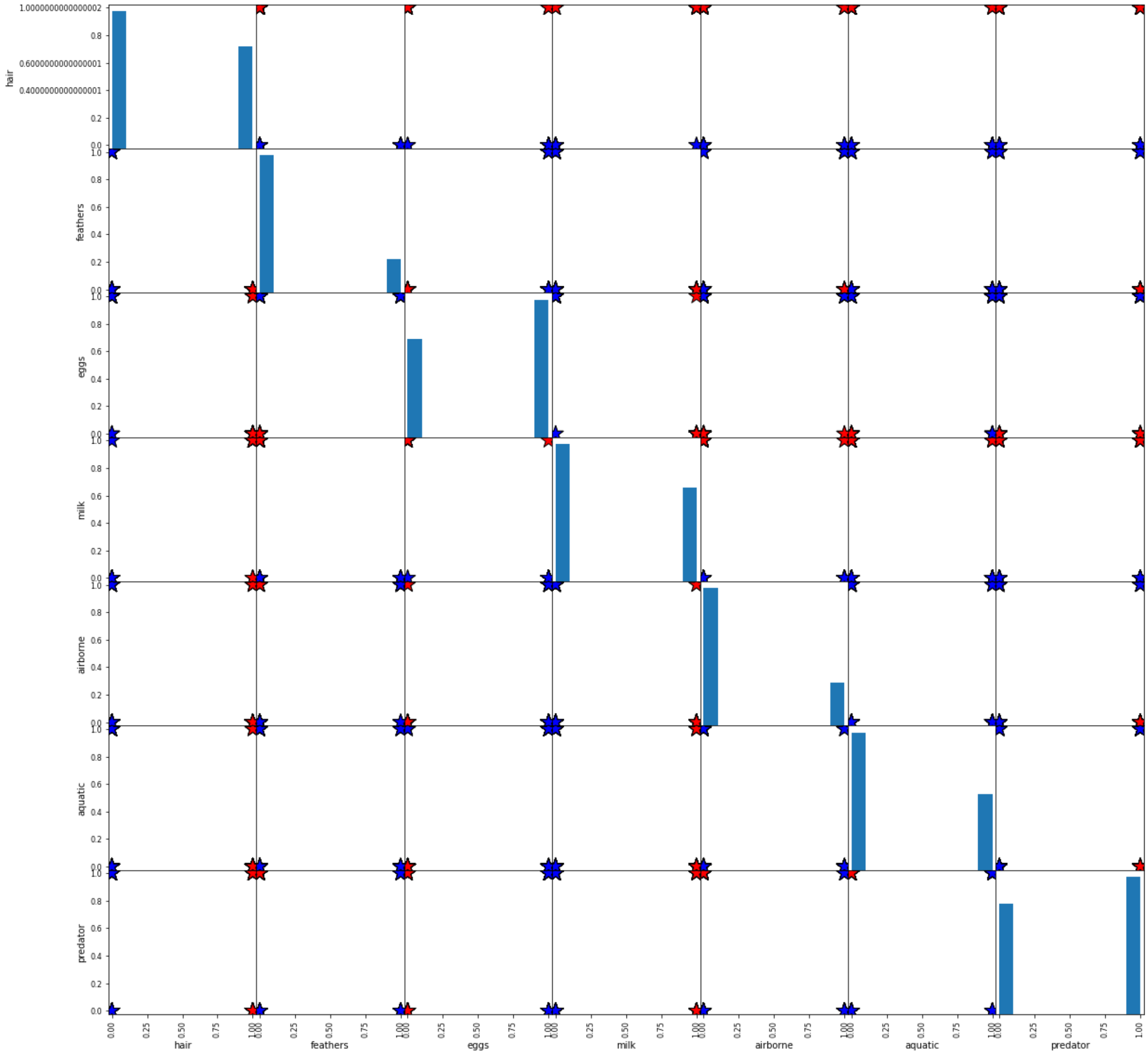
Out[29]:

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

Plotting scatter matrix

In [30]:

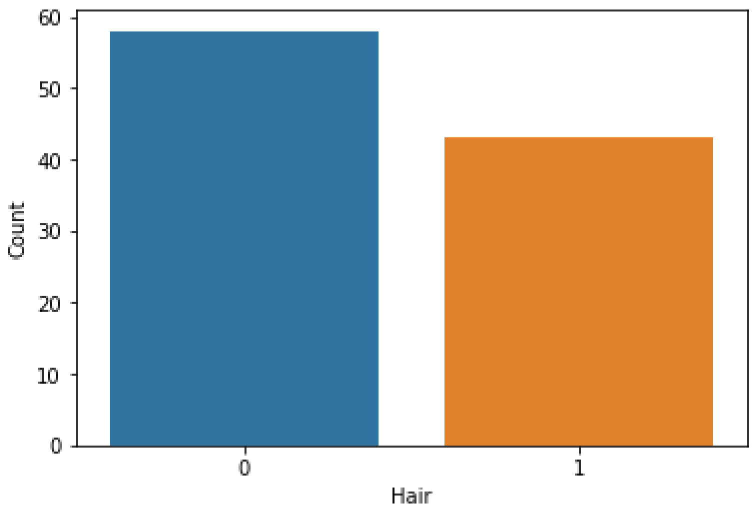
```
pd.plotting.scatter_matrix(zoo.iloc[:,:7],
                                       c=color_list,
                                       figsize= [20,20],
                                       diagonal='hist',
                                       alpha=1,
                                       s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



Visualizing has hair or not ?

In [31]:

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



Out[31]:

```
0    58
1    43
Name: hair, dtype: int64
```

KNN

In [32]:

```
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]
```

Train Test Split

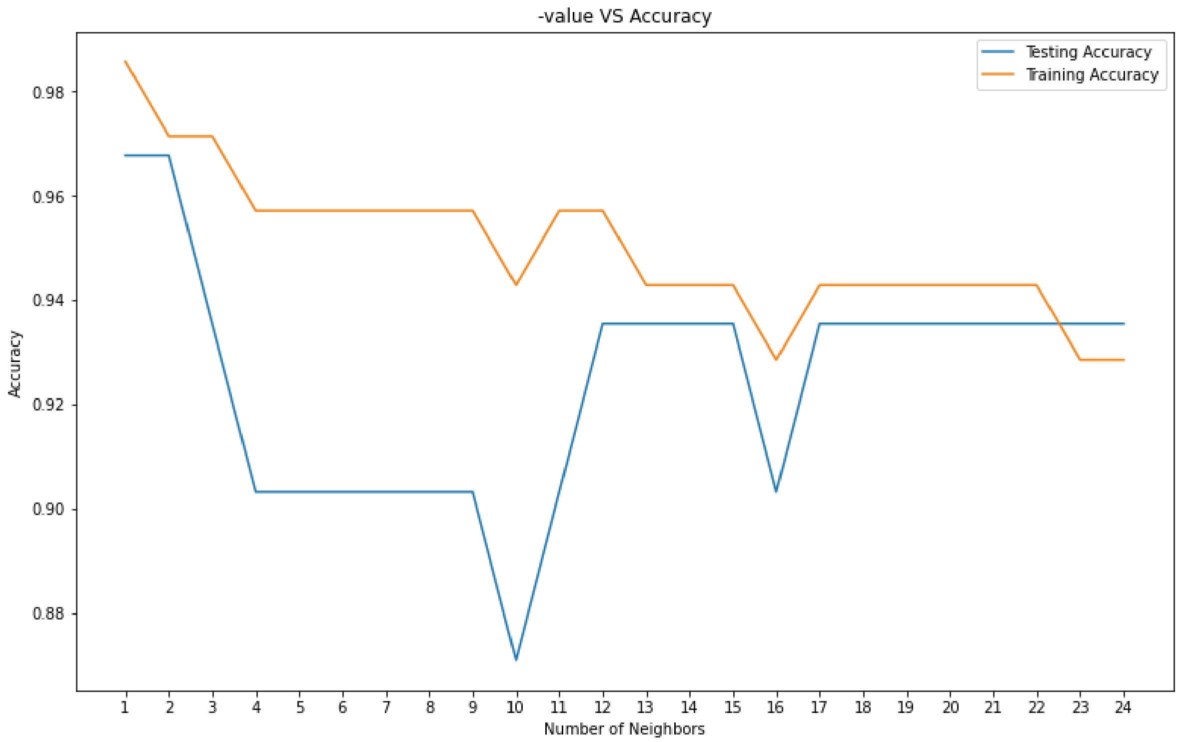
In [33]:

```
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 [34]:

```
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.index(
```

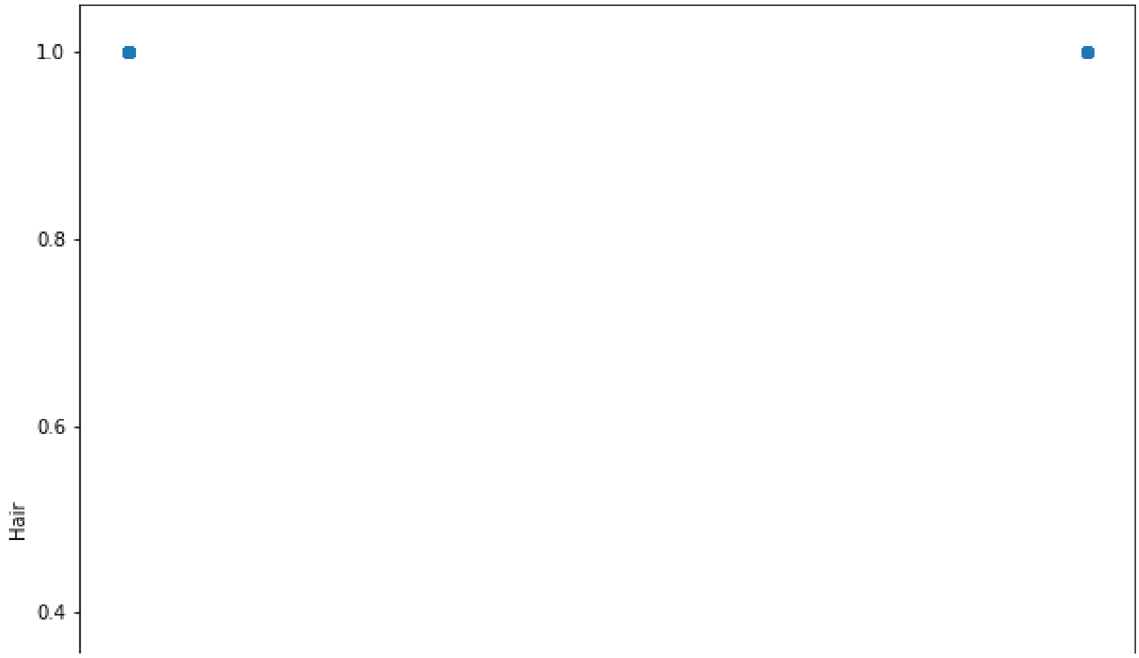



```
Best accuracy is 0.967741935483871 with K = 1
```

Visualizing Eggs and Hair on Scatter

In [35]:

```
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()
```

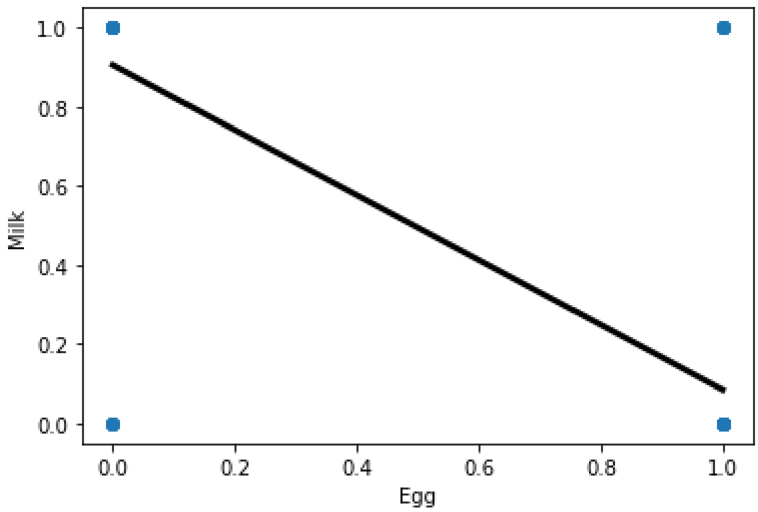


Linear Regression

In [36]:

```
# Plotting regression line and scatter
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
```



Cross Validation

In [37]:

```
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
```

Ridge

In [38]:

```
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
```

Lasso

In [39]:

```
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.          0.998
30154 -0.        ]
```

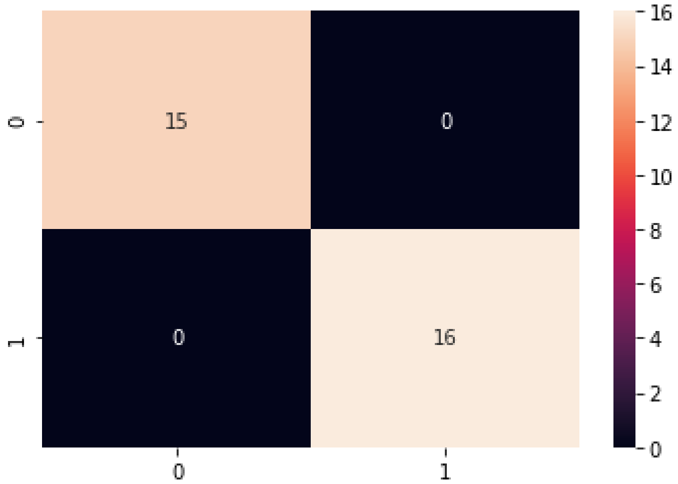
In [40]:

```
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 [41]:

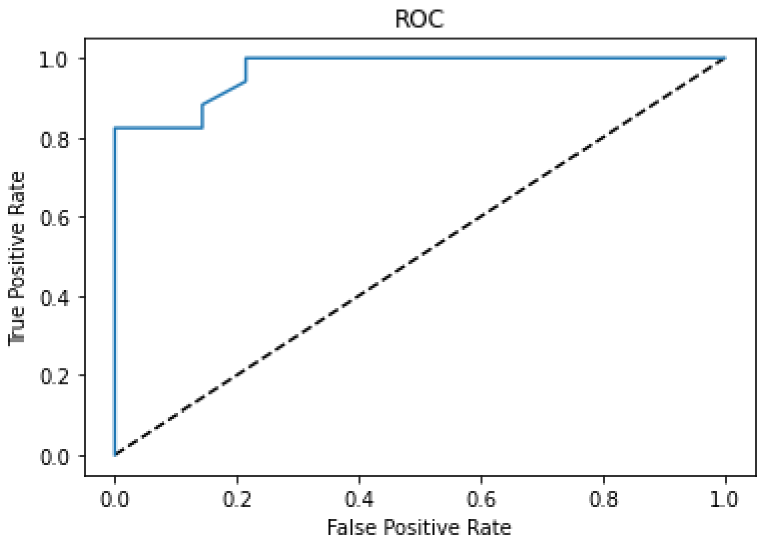
```
sns.heatmap(cm,annot=True,fmt="d")
plt.show()
```



Logistic Regression

In [42]:

```
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 [43]:

```
# grid search cross validation with 1 hyperparameter
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 [44]:

```
# grid search cross validation with 2 hyperparameter
# 1. hyperparameter is C:logistic regression regularization parameter
# 2. penalty l1 or l2
# Hyperparameter grid
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 = 12)
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_))
```

```
  File "C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\linear_model\_
logistic.py", line 443, in _check_solver
    raise ValueError("Solver %s supports only 'l2' or 'none' penalties, "
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 pe
nalty.

  warnings.warn("Estimator fit failed. The score on this train-test"
C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\model_selection\_valid
ation.py:610: FitFailedWarning: Estimator fit failed. The score on this tr
ain-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\model_selectio
n\_validation.py", line 593, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\linear_model\_
logistic.py", line 1306, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\linear_model\_
logistic.py", line 443, in _check_solver
    raise ValueError("Solver %s supports only 'l2' or 'none' penalties, "
```

In [45]:

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

Out[45]:

	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venom
0	1	0	0	1	0	0	1	1	1	1	
1	1	0	0	1	0	0	0	1	1	1	
2	0	0	1	0	0	1	1	1	1	0	
3	1	0	0	1	0	0	1	1	1	1	
4	1	0	0	1	0	0	1	1	1	1	
5	1	0	0	1	0	0	0	1	1	1	
6	1	0	0	1	0	0	0	1	1	1	
7	0	0	1	0	0	1	0	1	1	0	
8	0	0	1	0	0	1	1	1	1	0	
9	1	0	0	1	0	0	0	1	1	1	

Support Vector Machine

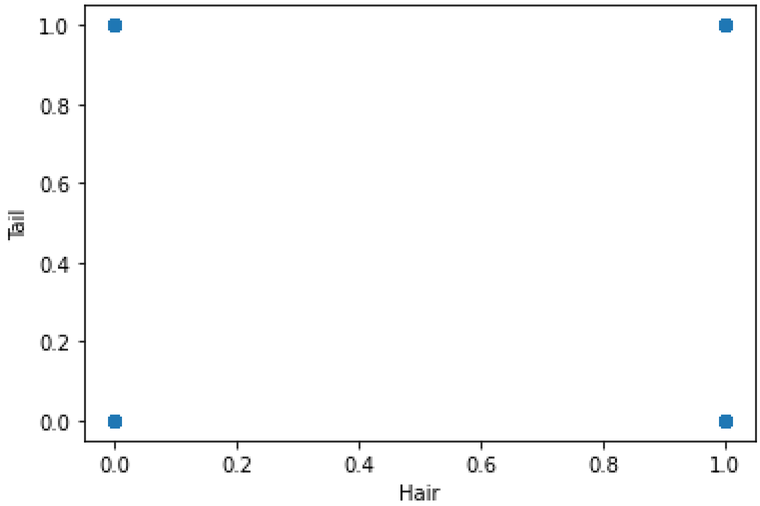
In [46]:

```
# SVM, pre-process and pipeline
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 [47]:

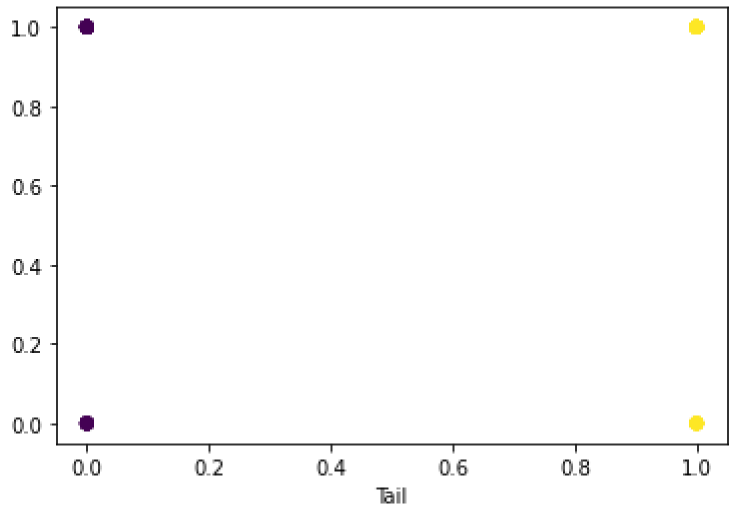
```
plt.scatter(zoo['hair'],zoo['tail'])
plt.xlabel('Hair')
plt.ylabel('Tail')
plt.show()
```



K-Means Clustering

In [48]:

```
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 [49]:

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

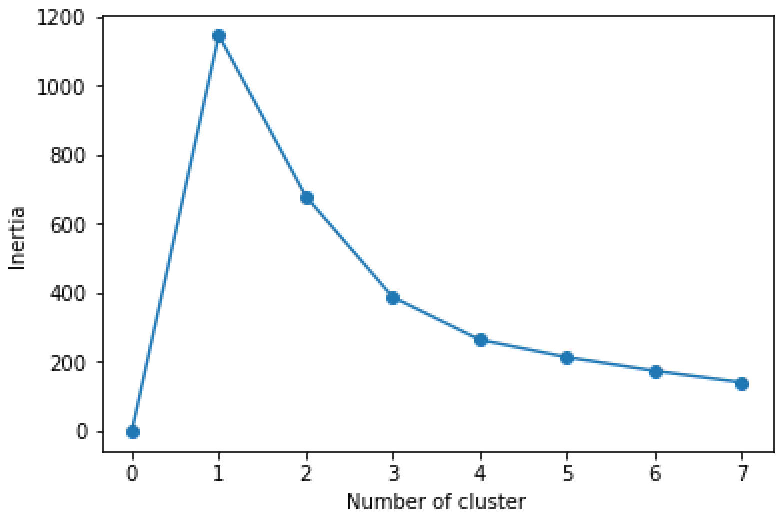
```
hair     0   1
labels        
0       58   0
1        0  43
```

Inertia

In [50]:

```
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()
# we choose the elbow < 1
```

```
C:\Users\LENOVO\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting th
e environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



In [51]:

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

In [52]:

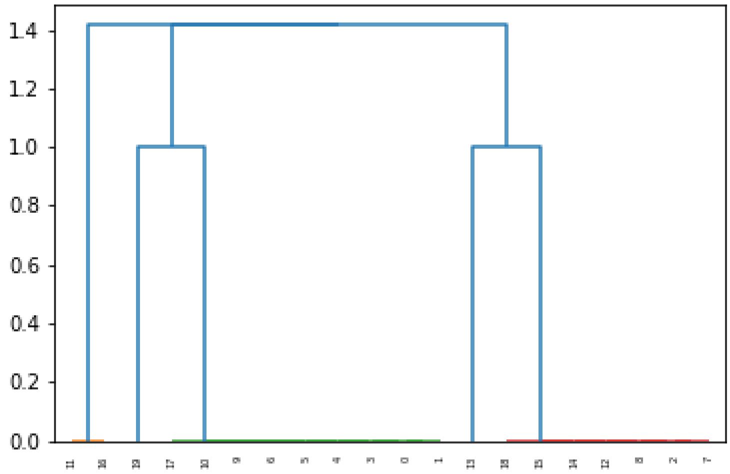
```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
scalar = StandardScaler()
kmeans = KMeans(n_clusters = 2)
pipe = make_pipeline(scalar,kmeans)
pipe.fit(data2)
labels = pipe.predict(data2)
df = pd.DataFrame({'labels':labels,"hair":zoo['hair']})
ct = pd.crosstab(df['labels'],df['hair'])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

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

Dendogram

In [53]:

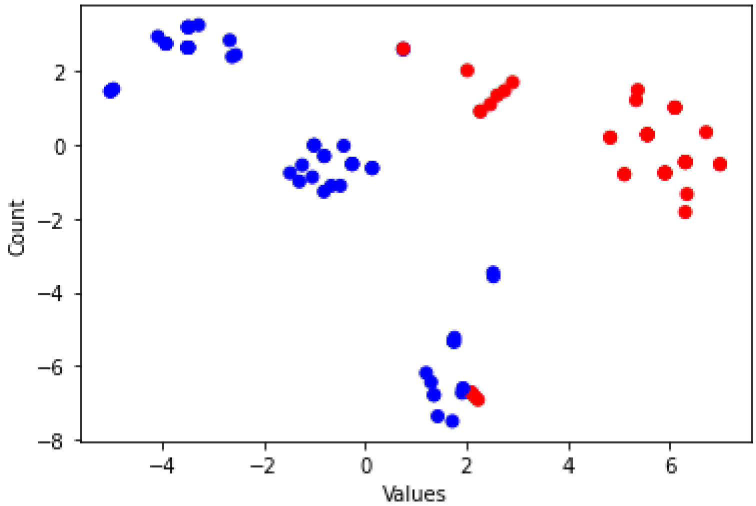
```
from scipy.cluster.hierarchy import linkage,dendrogram

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



t-distributed Stochastic Neighbor Embedding

In [54]:

```
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()
```



PCA

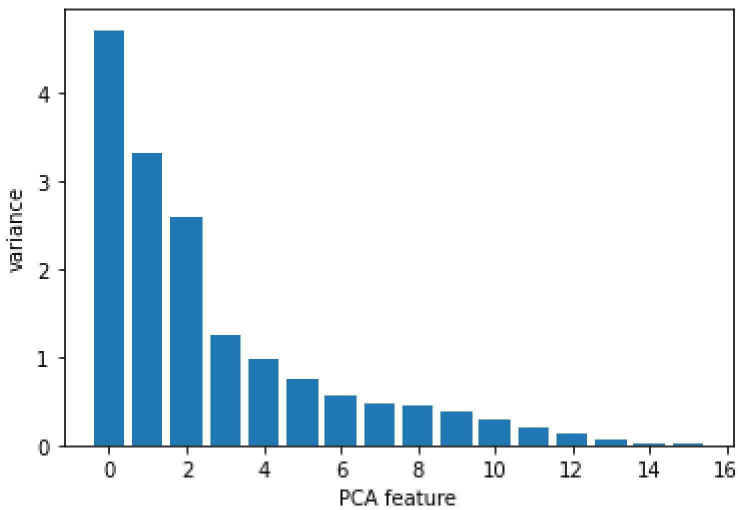
In [55]:

```
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]]
```

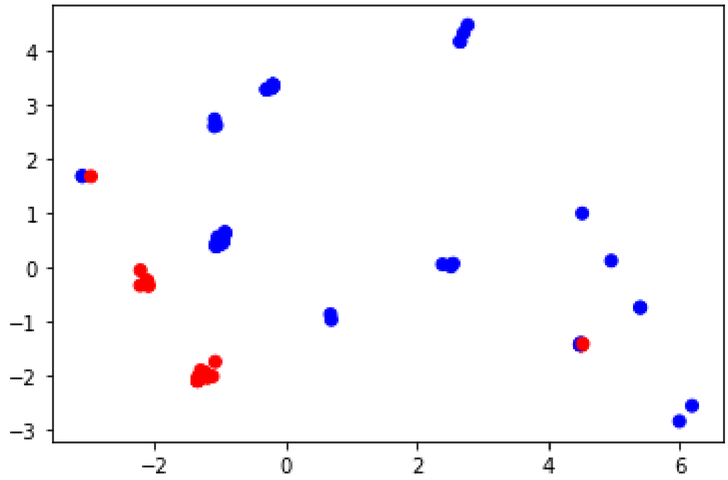
In [56]:

```
# PCA variance
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')
plt.ylabel('variance')
plt.show()
```



In [57]:

```
# apply PCA
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 [ ]:

```
```