In [1]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns # data visualization library
import matplotlib.pyplot as plt
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files i
import time
```

In [2]:

```
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectKBest
from sklearn.cross_validation import StratifiedKFold
from sklearn.grid_search import GridSearchCV
from sklearn.ensemble.gradient_boosting import GradientBoostingClassifier
from sklearn.cross_validation import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
import statsmodels.formula.api as sm
from sklearn.cross_validation import train_test_split
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.metrics import roc_curve, auc
```

```
C:\Users\march\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: D
eprecationWarning: This module was deprecated in version 0.18 in favor of th
e model_selection module into which all the refactored classes and functions
are moved. Also note that the interface of the new CV iterators are differen
t from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Users\march\Anaconda3\lib\site-packages\sklearn\grid_search.py:42: Deprec
ationWarning: This module was deprecated in version 0.18 in favor of the mod
el_selection module into which all the refactored classes and functions are
moved. This module will be removed in 0.20.
  DeprecationWarning)
```

In [3]:

```
data = pd.read_csv('C:\\Users\\march\A3TESTEOFICIAL.csv', sep=';')
```

In [4]:

```
data.head()
```

Out[4]:

	Categoria	Area do animal em pixels	Area do animal em cm2	Coordenadas em X	Coordenadas em Y	Altura em pixels	Altura em cm	Largura em pixels	Largura em cm
0	Immobility	1984.0	36.2709	199.811	137.286	34.5949	4.67757	111.041	15.0138
1	Immobility	1984.0	36.2709	199.811	137.286	34.5949	4.67757	111.041	15.0138
2	Immobility	2033.0	37.1667	197.768	139.548	33.6169	4.54534	112.922	15.2682
3	Immobility	2048.5	37.4501	196.501	140.938	39.7564	5.37546	106.804	14.4409
4	Immobility	2048.5	37.4501	196.493	140.941	39.7564	5.37546	106.804	14.4409

In [5]:

```
# Well know question is is there any NaN value and length of this data so lets look at info
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4982 entries, 0 to 4981
Data columns (total 15 columns):
Categoria                                          4982 non-null object
Area do animal em pixels                           4982 non-null float64
Area do animal em cm2                              4982 non-null float64
Coordenadas em X                                   4982 non-null float64
Coordenadas em Y                                   4982 non-null float64
Altura em pixels                                   4982 non-null float64
Altura em cm                                       4982 non-null float64
Largura em pixels                                  4982 non-null float64
Largura em cm                                      4982 non-null float64
Angulo entre o animal e a coordenada horizontal    4982 non-null float64
Variacao da area em pixels                         4982 non-null float64
Distancia percorrida pela centroide                4982 non-null float64
Variacao do comprimento em pixels                  4982 non-null float64
Variacao da largura em pixels                      4982 non-null float64
Variacao do angulo do animal em pixels             4982 non-null float64
dtypes: float64(14), object(1)
memory usage: 583.9+ KB
```

In [6]:

```
data.describe()
```

Out[6]:

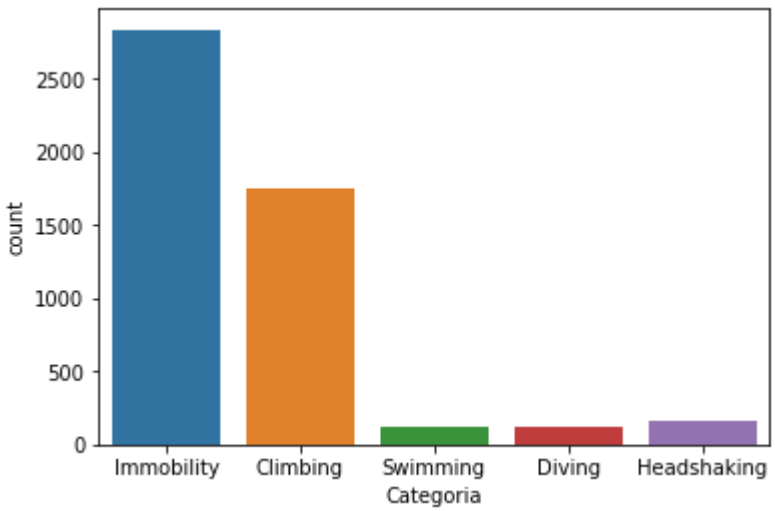
	Area do animal em pixels	Area do animal em cm2	Coordenadas em X	Coordenadas em Y	Altura em pixels	Altura em cm	Largu
count	4982.000000	4982.000000	4982.000000	4982.000000	4982.000000	4982.000000	4982.0
mean	865.730128	15.827025	186.429435	156.572927	37.285550	5.041374	37.4
std	458.677155	8.385402	26.788891	24.761682	19.268225	2.605254	19.1
min	0.500000	0.009141	106.667000	69.000000	0.894427	0.120935	0.7
25%	533.125000	9.746435	165.485250	143.660250	22.313300	3.016970	25.7
50%	923.000000	16.874000	191.144000	162.448500	31.418700	4.248125	30.7
75%	1196.875000	21.880925	207.329500	173.409750	55.066325	7.445515	52.7
max	2219.500000	40.576300	239.087000	198.377000	102.915000	13.915100	112.9

In [7]:

```
sns.countplot(x="Categoria", data=data)
data.loc[:,'Categoria'].value_counts()
```

Out[7]:

```
Immobility     2835
Climbing       1747
Headshaking     155
Diving          126
Swimming        119
Name: Categoria, dtype: int64
```



In [8]:

```
# KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3)
x,y = data.loc[:,data.columns != 'Categoria'], data.loc[:,'Categoria']
knn.fit(x,y)
prediction = knn.predict(x)
print('Prediction: {}'.format(prediction))
```

```
Prediction: ['Immobility' 'Immobility' 'Immobility' ... 'Climbing' 'Climbin
g'
 'Climbing']
```

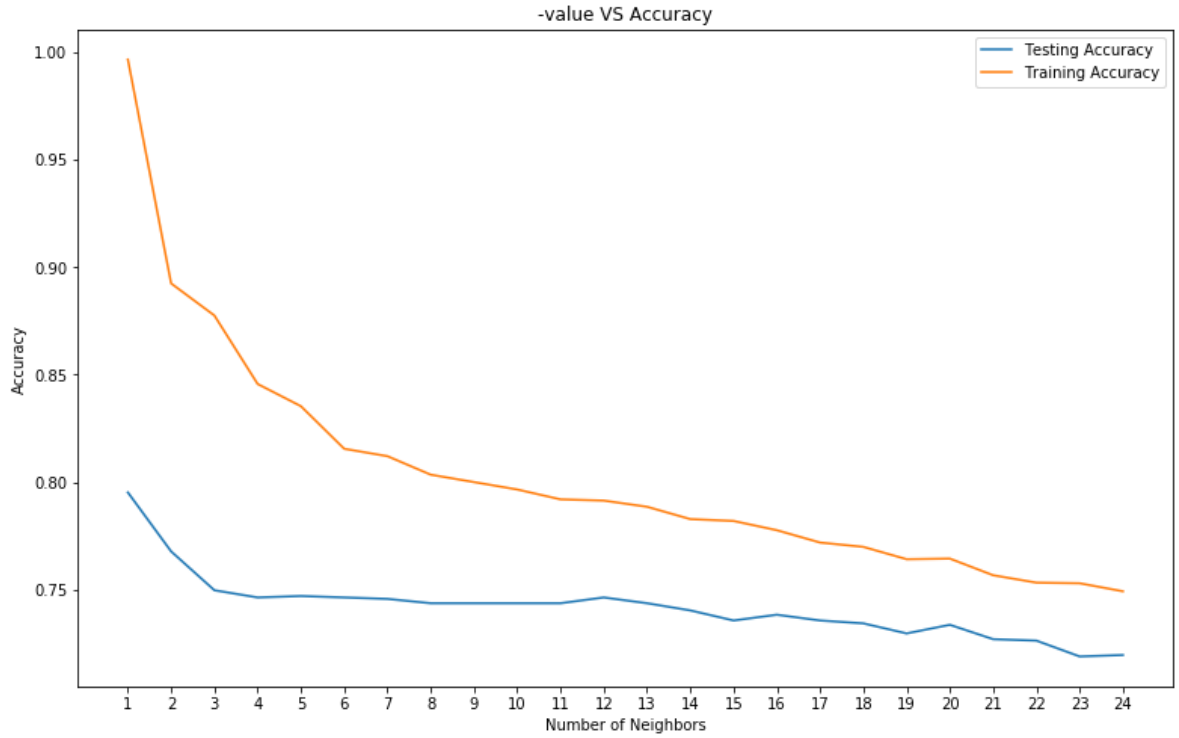
In [9]:

```
# train test split
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 = 3)
x,y = data.loc[:,data.columns != 'Categoria'], data.loc[:,'Categoria']
knn.fit(x_train,y_train)
prediction = knn.predict(x_test)
#print('Prediction: {}'.format(prediction))
print('With KNN (K=3) accuracy is: ',knn.score(x_test,y_test)) # accuracy
```

```
With KNN (K=3) accuracy is:  0.7498327759197324
```

In [10]:

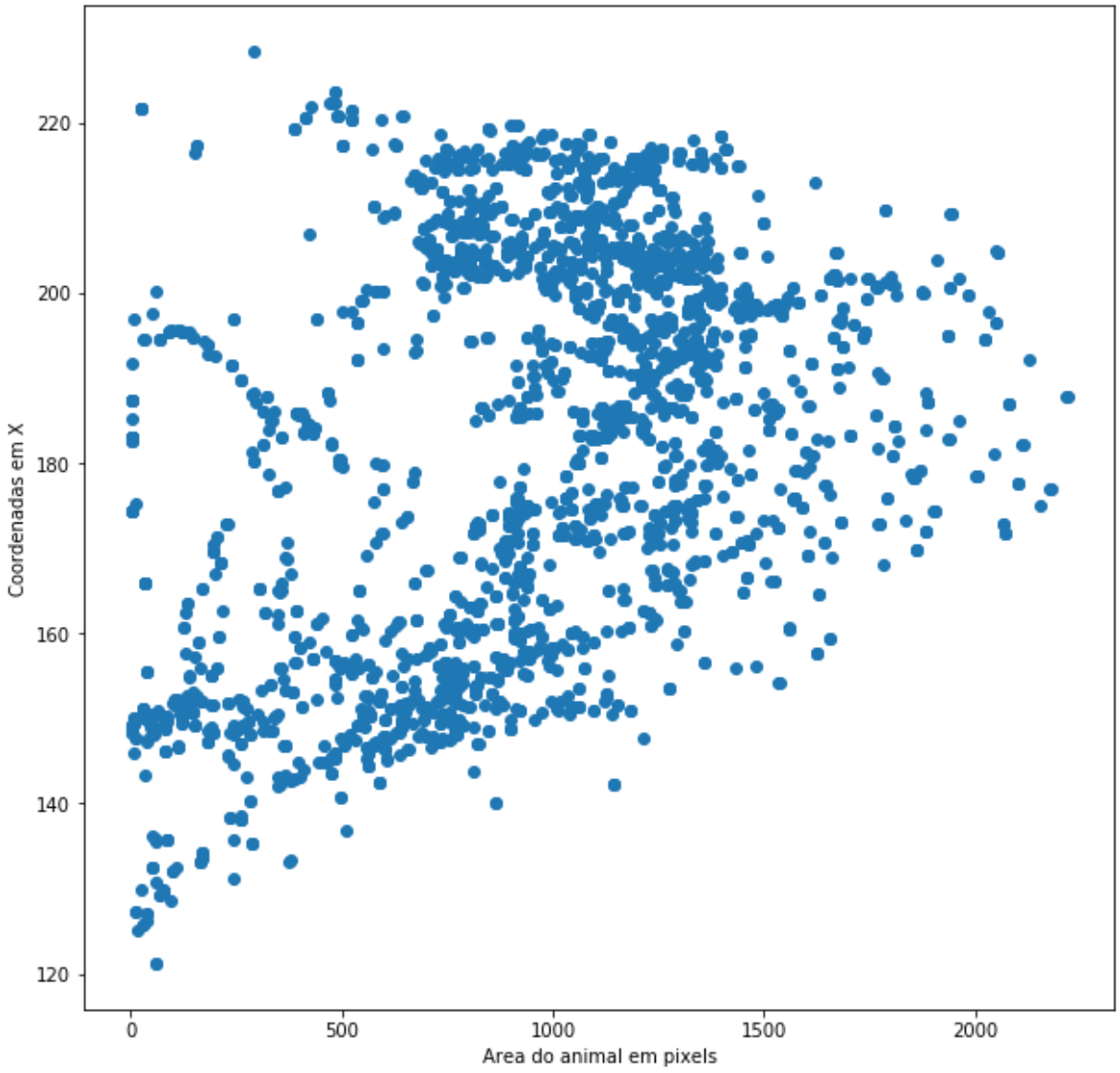
```
# Model complexity
neig = np.arange(1, 25)
train_accuracy = []
test_accuracy = []
# Loop over different values of k
for i, k in enumerate(neig):
    # k from 1 to 25(exclude)
    knn = KNeighborsClassifier(n_neighbors=k)
    # Fit with knn
    knn.fit(x_train,y_train)
    #train accuracy
    train_accuracy.append(knn.score(x_train, y_train))
    # test accuracy
    test_accuracy.append(knn.score(x_test, y_test))

# Plot
plt.figure(figsize=[13,8])
plt.plot(neig, test_accuracy, label = 'Testing Accuracy')
plt.plot(neig, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.title('-value VS Accuracy')
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.xticks(neig)
plt.savefig('graph.png')
plt.show()
print("Best accuracy is {} with K = {}".format(np.max(test_accuracy),1+test_accuracy.index(
```




```
Best accuracy is 0.7953177257525084 with K = 1
```

In [11]:

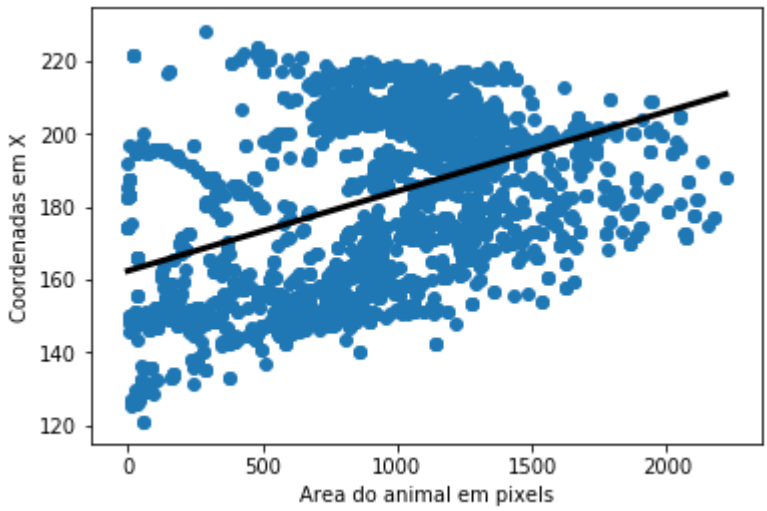
```
data1 = data[data['Categoria'] =='Immobility']
x = np.array(data1.loc[:,'Area do animal em pixels']).reshape(-1,1)
y = np.array(data1.loc[:,'Coordenadas em X']).reshape(-1,1)
# Scatter
plt.figure(figsize=[10,10])
plt.scatter(x=x,y=y)
plt.xlabel('Area do animal em pixels')
plt.ylabel('Coordenadas em X')
plt.show()
```



In [12]:

```
# LinearRegression
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
# Predict space
predict_space = np.linspace(min(x), max(x)).reshape(-1,1)
# Fit
reg.fit(x,y)
# Predict
predicted = reg.predict(predict_space)
# R^2
print('R^2 score: ',reg.score(x, y))
# Plot regression line and scatter
plt.plot(predict_space, predicted, color='black', linewidth=3)
plt.scatter(x=x,y=y)
plt.xlabel('Area do animal em pixels')
plt.ylabel('Coordenadas em X')
plt.show()
```

```
R^2 score:  0.17588025476025215
```



In [13]:

```
# CV
from sklearn.model_selection import cross_val_score
reg = LinearRegression()
k = 3
cv_result = cross_val_score(reg,x,y,cv=k) # uses R^2 as score
print('CV Scores: ',cv_result)
print('CV scores average: ',np.sum(cv_result)/k)
```

```
CV Scores:  [-0.66954863  0.17289111  0.07587535]
CV scores average:  -0.14026072137940535
```

In [14]:

```
# Ridge
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 = 0.1, 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:  0.17075563183238507
```

In [15]:

```
# Lasso
from sklearn.linear_model import Lasso
x = np.array(data1.loc[:,['Area do animal em pixels','Coordenadas em X','Coordenadas em Y',
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state = 3, test_size = 0.3)
lasso = Lasso(alpha = 0.1, 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:  0.9624420167633475
Lasso coefficients:  [ 0.          0.80623246 -0.          0.         -0.
     ]
```

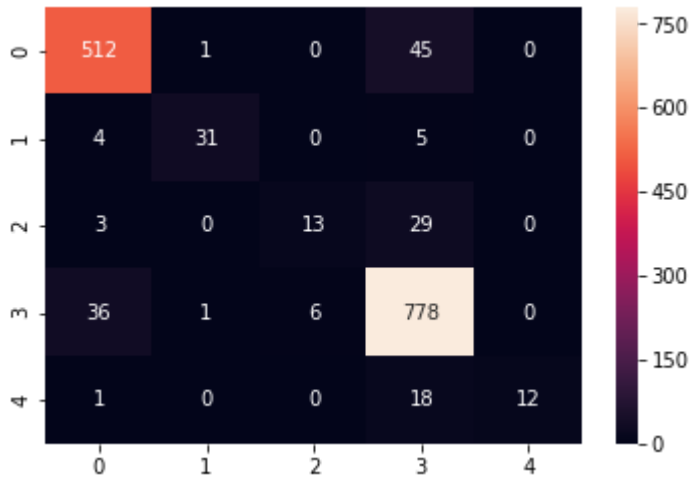
In [16]:

```
# Confusion matrix with random forest
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
x,y = data.loc[:,data.columns != 'Categoria'], data.loc[:,'Categoria']
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('Confusion matrix: \n',cm)
print('Classification report: \n',classification_report(y_test,y_pred))
```

```
Confusion matrix:
 [[512   1   0  45   0]
 [  4  31   0   5   0]
 [  3   0  13  29   0]
 [ 36   1   6 778   0]
 [  1   0   0  18  12]]
Classification report:
              precision    recall  f1-score   support

   Climbing       0.92      0.92      0.92       558
     Diving       0.94      0.78      0.85        40
Headshaking       0.68      0.29      0.41        45
 Immobility       0.89      0.95      0.92       821
   Swimming       1.00      0.39      0.56        31

avg / total       0.90      0.90      0.89      1495
```

In [17]:

```
# visualize with seaborn library
sns.heatmap(cm,annot=True,fmt="d")
plt.show()
```



In [18]:

```
# 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': 15}
Best score: 0.6601766358892012
```

In [19]:

```
# 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_))
```

```
Tuned hyperparameters : {'C': 100.0, 'penalty': 'l1'}
Best Accuracy: 0.7542299971322053
```

In [20]:

```
#Etapa de pre processamento dos dados
# Load data
data = pd.read_csv('C:\\Users\\march\A3TESTEOFICIAL.csv', sep=';')
# get_dummies
df = pd.get_dummies(data)
df.head(10)
```

Out[20]:

	Area do animal em pixels	Area do animal em cm2	Coordenadas em X	Coordenadas em Y	Altura em pixels	Altura em cm	Largura em pixels	Largura em cm	Angul entre animal e coordenad horizonta
0	1984.0	36.2709	199.811	137.286	34.5949	4.67757	111.041	15.0138	-25.560
1	1984.0	36.2709	199.811	137.286	34.5949	4.67757	111.041	15.0138	-25.560
2	2033.0	37.1667	197.768	139.548	33.6169	4.54534	112.922	15.2682	-28.523
3	2048.5	37.4501	196.501	140.938	39.7564	5.37546	106.804	14.4409	-21.447
4	2048.5	37.4501	196.493	140.941	39.7564	5.37546	106.804	14.4409	-21.447
5	2022.0	36.9656	194.535	143.105	38.8598	5.25423	109.851	14.8529	-31.522
6	2022.0	36.9656	194.535	143.105	38.8598	5.25423	109.851	14.8529	-31.522
7	2022.0	36.9656	194.535	143.105	38.8598	5.25423	109.851	14.8529	-31.522
8	2128.5	38.9126	192.237	144.801	35.7135	4.82882	107.840	14.5810	-29.981
9	2219.5	40.5763	187.935	146.530	39.3308	5.31791	109.883	14.8572	-29.320

In [23]:

```
y = data.Categoria
list = ['Categoria']
x = data.drop(list,axis = 1 )
```

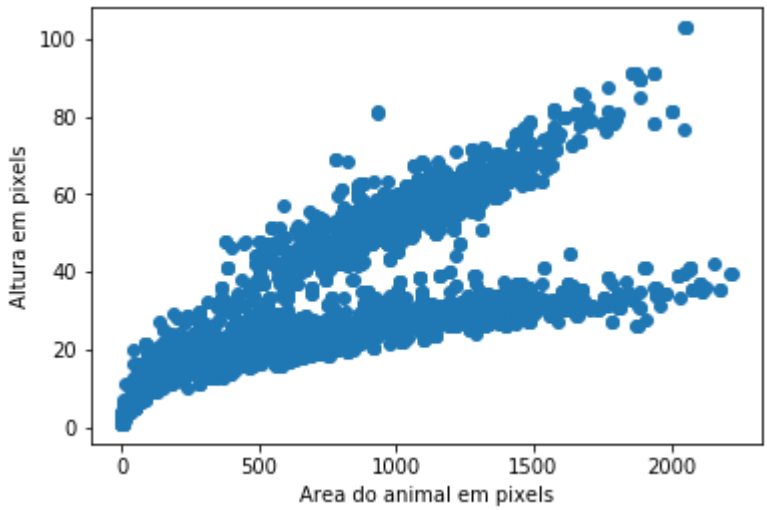
In [24]:

```
# 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: 0.8645937813440321
Tuned Model Parameters: {'SVM__C': 100, 'SVM__gamma': 0.1}
```

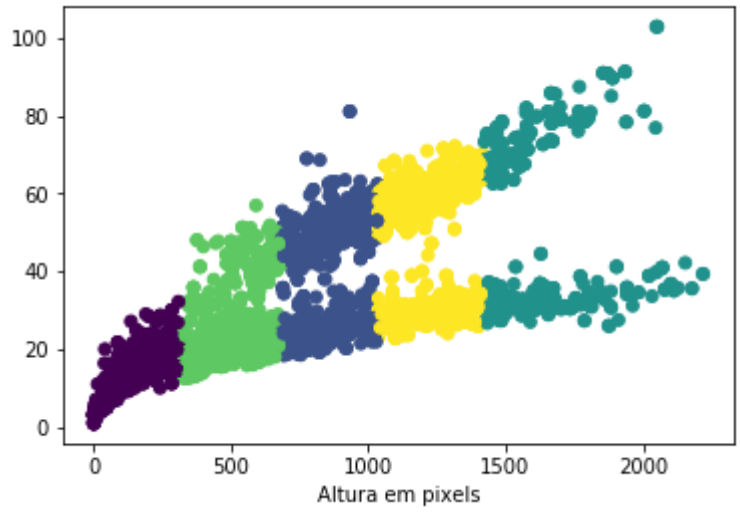
In [25]:

```
#Unsupervised learnig
# As you can see there is no labels in data
data = pd.read_csv('C:\\Users\\march\A3TESTEOFICIAL.csv', sep=';')
plt.scatter(data['Area do animal em pixels'],data['Altura em pixels'])
plt.xlabel('Area do animal em pixels')
plt.ylabel('Altura em pixels')
plt.show()
```



In [26]:

```
# KMeans Clustering
data2 = data.loc[:,['Area do animal em pixels','Altura em pixels']]
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 5)
kmeans.fit(data2)
labels = kmeans.predict(data2)
plt.scatter(data['Area do animal em pixels'],data['Altura em pixels'],c = labels)
plt.xlabel('Area do animal em pixels')
plt.xlabel('Altura em pixels')
plt.show()
```



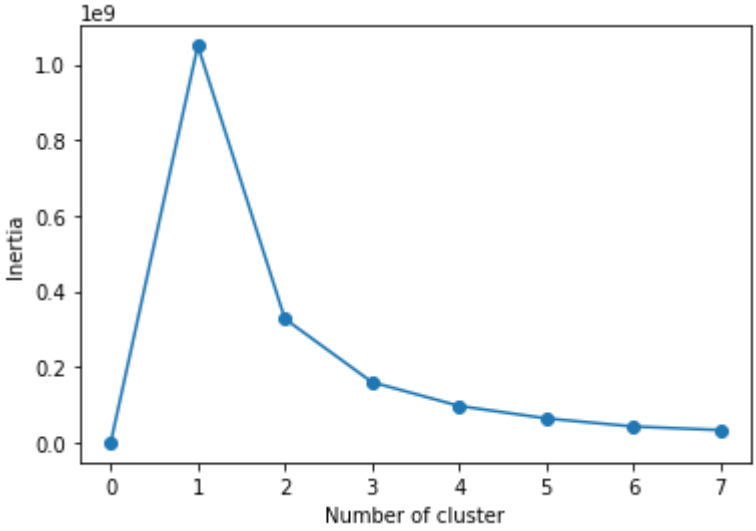
In [27]:

```
# cross tabulation table
#avaliacao do clustering
df = pd.DataFrame({'labels':labels,"Categoria":data['Categoria']})
ct = pd.crosstab(df['labels'],df['Categoria'])
print(ct)
```

```
Categoria  Climbing  Diving  Headshaking  Immobility  Swimming
labels
0               406      95            0         305         0
1               494       8           73         791         4
2                62       4            6         335        31
3               382      16            7         360         0
4               403       3           69        1044        84
```

In [28]:

```
# inertia
inertia_list = np.empty(8)
for i in range(1,8):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data2)
    inertia_list[i] = kmeans.inertia_
plt.plot(range(0,8),inertia_list,'-o')
plt.xlabel('Number of cluster')
plt.ylabel('Inertia')
plt.show()
```



In [29]:

```
data = pd.read_csv('C:\\Users\\march\A3TESTEOFICIAL.csv', sep=';')
data3 = data.drop('Categoria',axis = 1)
```

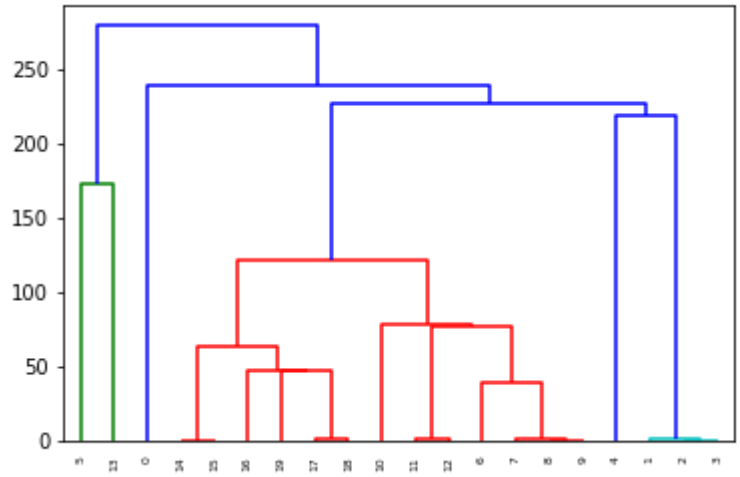
In [30]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
scalar = StandardScaler()
kmeans = KMeans(n_clusters = 5)
pipe = make_pipeline(scalar,kmeans)
pipe.fit(data3)
labels = pipe.predict(data3)
df = pd.DataFrame({'labels':labels,"Categoria":data['Categoria']})
ct = pd.crosstab(df['labels'],df['Categoria'])
print(ct)
```

```
Categoria  Climbing  Diving  Headshaking  Immobility  Swimming
labels
0                37      10            0          24         0
1               419      29            7         402         0
2               615       3          113        1258        60
3               295      72            0         233         0
4               381      12           35         918        59
```

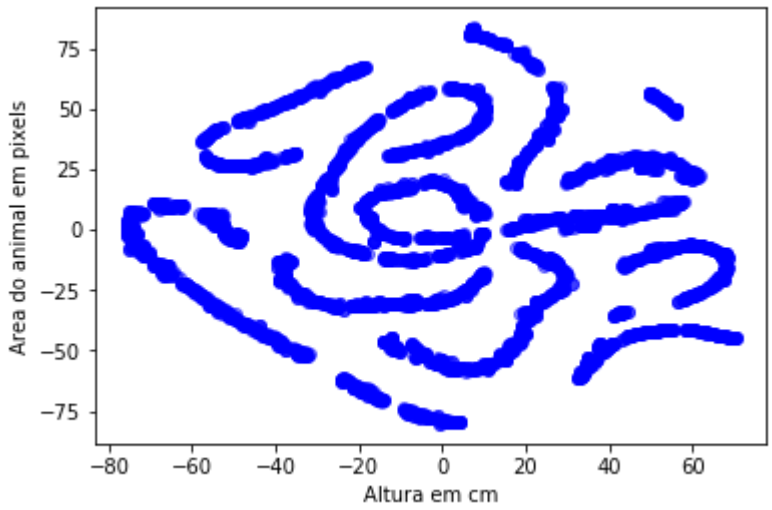
In [31]:

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

merg = linkage(data3.iloc[200:220,:],method = 'single')
dendrogram(merg, leaf_rotation = 90, leaf_font_size = 6)
plt.show()
```



In [36]:

```
from sklearn.manifold import TSNE
model = TSNE(learning_rate=100)
transformed = model.fit_transform(data2)
x = transformed[:,0]
y = transformed[:,1]
plt.scatter(x, y, c='blue', alpha=0.5 )
plt.ylabel('Area do animal em pixels')
plt.xlabel('Altura em cm')
plt.show()
```



In [33]:

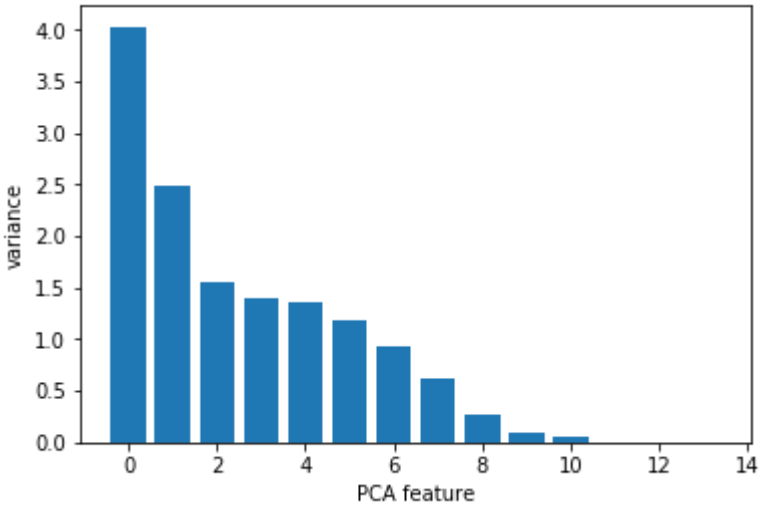
```
# PCA
from sklearn.decomposition import PCA
model = PCA()
model.fit(data3)
transformed = model.transform(data3)
print('Principle components: ',model.components_)
```

```
Principle components:  [[ 9.98651274e-01  1.82570510e-02  1.77006257e-02  1.
67575010e-02
   2.99844863e-02  4.05419824e-03  2.38634514e-02  3.22657246e-03
  -3.09603547e-03  1.60966222e-02 -1.78539686e-03  2.19816428e-04
   9.28319505e-04 -1.44253808e-04]
 [-1.61101573e-02 -2.94528859e-04 -5.55082558e-03  5.29141728e-04
   1.15275576e-02  1.55863724e-03 -1.14349427e-02 -1.54611460e-03
   4.09934232e-03  9.98867897e-01 -4.73892343e-03  2.65749633e-02
   3.05636109e-02 -4.91143068e-03]
 [-1.93576485e-03 -3.53836300e-05  7.43034376e-01 -6.35866903e-01
   7.80624655e-02  1.05548164e-02 -1.02256881e-01 -1.38261387e-02
   1.54578872e-01  2.03961641e-03  2.29210177e-03  2.79716442e-03
  -4.99871105e-03  5.27239191e-02]
 [-2.79798690e-03 -5.11361195e-05  1.24692480e-01 -1.22810899e-01
  -5.95614987e-02 -8.05330245e-03  6.49145184e-02  8.77708015e-03
  -8.87502310e-01  3.57253712e-03 -6.04742077e-03 -1.04260688e-02
   1.43888211e-02 -4.16452461e-01]
 [ 9.80039360e-03  1.79189760e-04 -1.50732370e-01 -3.38254842e-01
  -5.91562953e-01 -7.99851306e-02  6.78286537e-01  9.17109822e-02
   4.19162108e-02  1.45412764e-02  4.04750180e-03 -6.47810803e-02
   7.15811692e-02  1.63199890e-01]
 [ 3.81569714e-03  6.97600581e-05 -6.64358312e-02 -6.78695815e-02
  -6.65585750e-02 -8.99937555e-03  7.24524269e-02  9.79626814e-03
   4.29239683e-01 -5.11656420e-03 -1.69976546e-02 -3.06147853e-02
   3.46891348e-02 -8.91294944e-01]
 [ 2.56527731e-02  4.69003872e-04 -6.28999849e-01 -6.71679476e-01
   1.49556197e-01  2.02214598e-02 -3.19116360e-01 -4.31476855e-02
  -4.66642207e-02 -6.95080323e-03 -4.62815032e-03  9.77850469e-02
  -1.15569472e-01  3.01520875e-02]
 [ 5.21024483e-03  9.53000611e-05 -7.21332621e-02 -6.23344487e-02
   2.53880774e-02  3.43273561e-03 -1.69094174e-01 -2.28631787e-02
  -1.43420042e-02 -6.11216804e-03  1.03823196e-03 -6.74091289e-01
   7.10643636e-01  3.81389252e-02]
 [ 3.16445200e-02  5.78538004e-04  5.23585003e-02  6.70168082e-02
  -6.58545859e-01 -8.90418438e-02 -5.21350116e-01 -7.04917113e-02
   5.13824126e-03  9.84556977e-03  5.04235977e-01 -4.16571436e-02
  -1.30778235e-01 -1.28054868e-02]
 [-1.62627918e-02 -2.97324580e-04 -3.54635894e-02 -4.14429840e-02
   3.84349044e-01  5.19677362e-02  3.00244736e-01  4.05960557e-02
  -1.55920821e-03 -1.14692526e-03  8.63188713e-01  3.88915422e-02
   8.79774287e-02 -1.37619321e-02]
 [-5.36070263e-03 -9.77762685e-05 -5.52395992e-03 -6.27516749e-03
   1.12151858e-01  1.51639766e-02  9.47259857e-02  1.28079791e-02
  -1.69816262e-03  3.95645022e-02  1.60017296e-02 -7.25833653e-01
  -6.70292566e-01 -2.32705097e-03]
 [ 1.82762763e-02 -9.99702839e-01 -1.75021951e-08 -3.34404674e-08
   8.93530004e-04 -6.60848317e-03  1.96807862e-03 -1.45556994e-02
  -1.35166972e-08  1.47126013e-09  3.26701005e-09 -1.98177361e-07
  -1.23705685e-07 -2.88789996e-09]
 [-2.67900895e-04  1.46538346e-02  7.83393226e-09  1.72871193e-08
  -6.43418784e-04  4.75914507e-03  1.33974677e-01 -9.90864731e-01
  -5.66500472e-09  4.91993540e-09 -1.00878960e-08 -7.26492428e-08
```

```
  -4.24428887e-08  7.20379600e-09]
 [ 1.23165653e-04 -6.73723655e-03 -1.02392618e-08 -7.83555776e-09
  -1.33986029e-01  9.90949146e-01 -6.30268038e-04  4.66166614e-03
  -2.09501612e-09  3.82565435e-09  1.40186047e-08 -2.70251614e-08
  -4.48942682e-08 -2.86237872e-09]]
```

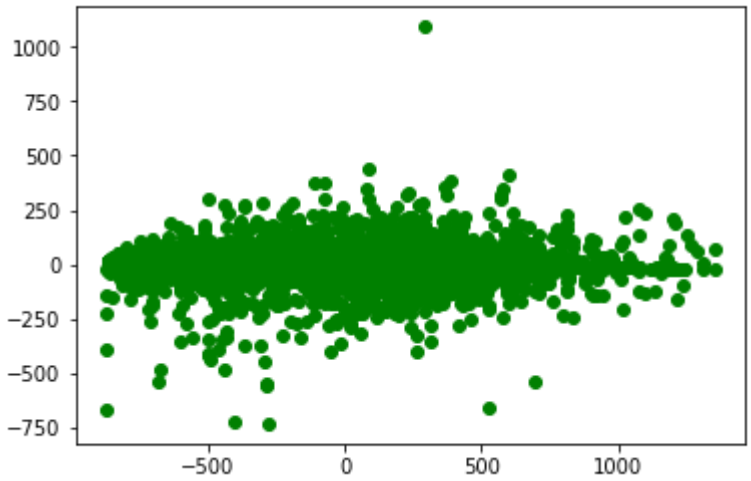
In [34]:

```
# PCA variance
scaler = StandardScaler()
pca = PCA()
pipeline = make_pipeline(scaler,pca)
pipeline.fit(data3)

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



In [37]:

```
# apply PCA
pca = PCA(n_components = 2)
pca.fit(data3)
transformed = pca.transform(data3)
x = transformed[:,0]
y = transformed[:,1]
plt.scatter(x,y,c = 'green')
plt.show()
```



In [ ]: