Implement a KNN model to classify the animals into categories

In [33]:
```
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import seaborn as sns
from sklearn.preprocessing import normalize, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix



from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, accuracy_score,precision_score,recall_score,f1_score,matthews
from sklearn.metrics import confusion_matrix

warnings.filterwarnings("ignore")
```

In [2]:
```
zoo = pd.read_csv("zoo.csv")
zoo.head()
```

Out[2]:

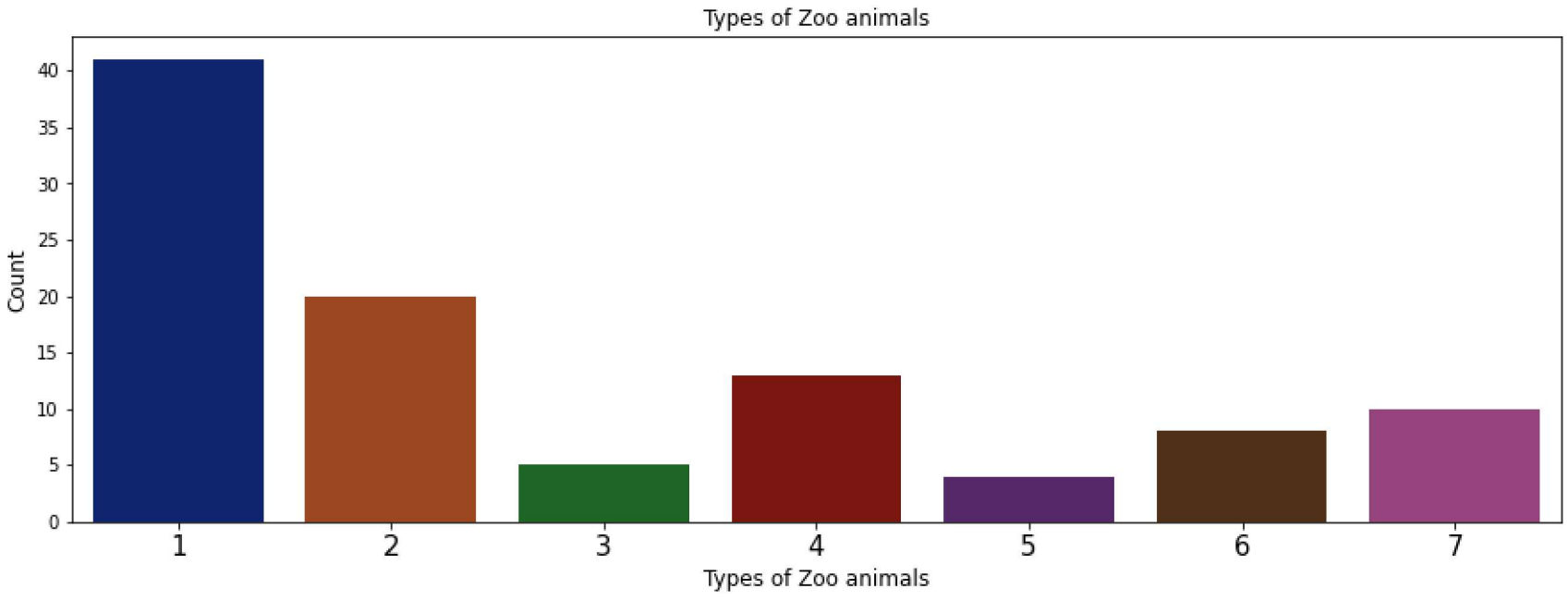
	animal name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs	tail	domestic
0	aardvark	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0
1	antelope	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0
2	bass	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
3	bear	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0
4	boar	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0

In [3]:
```
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
```

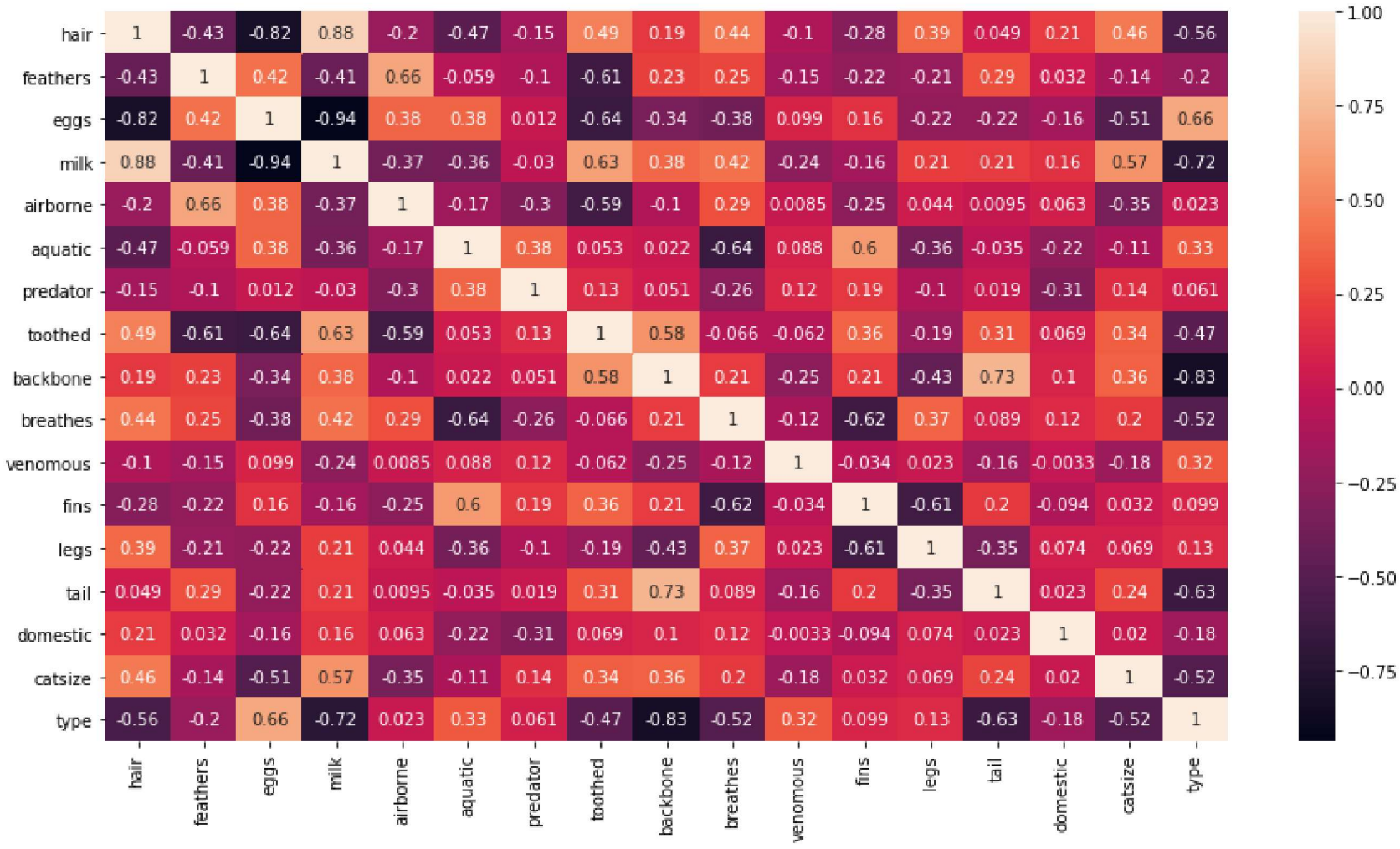
In [8]:
```
plt.figure(figsize=(15, 5))

plt.title("Types of Zoo animals")
sns.countplot(data=zoo, x="type",palette = "dark")
plt.xticks(rotation = 0, size = 15)
plt.xlabel("Types of Zoo animals", fontsize=12)
plt.ylabel("Count", fontsize=12)
```

Out[8]: Text(0, 0.5, 'Count')



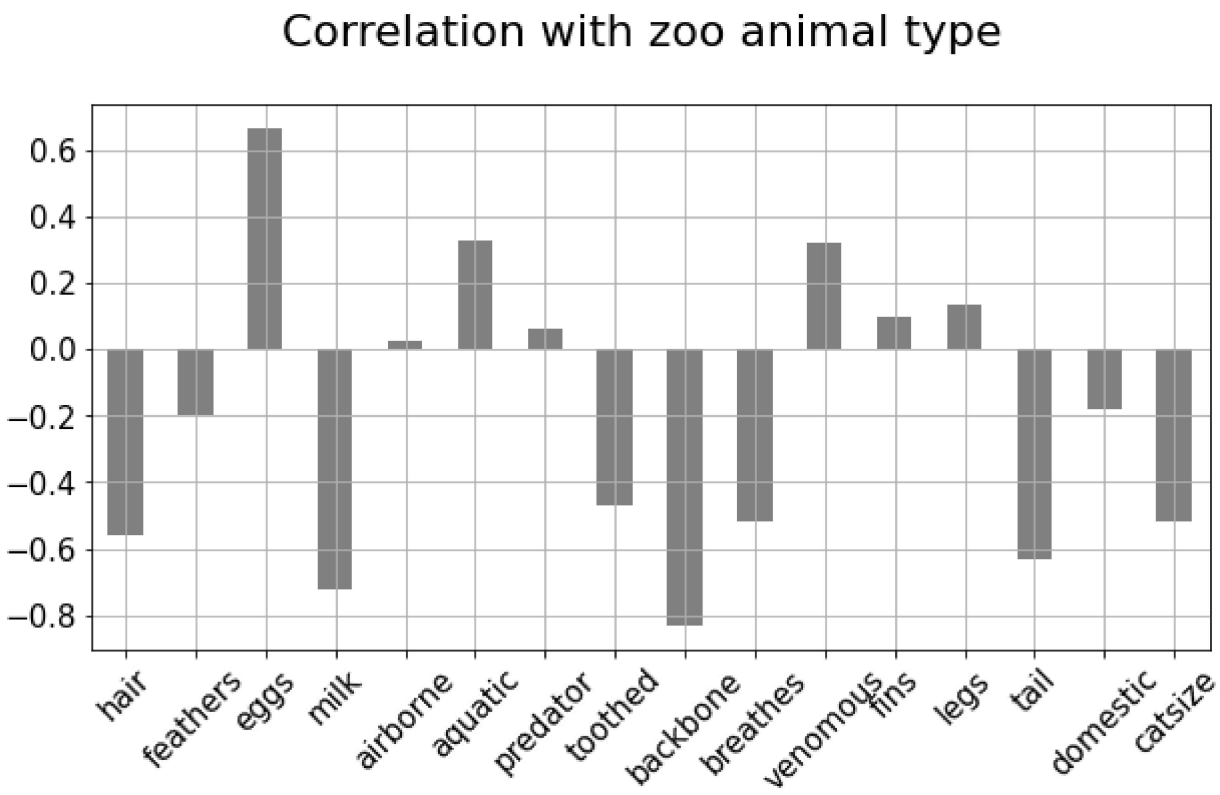
In [11]:
```
plt.figure(figsize=(15,8))
sns.heatmap(zoo.corr(),annot=True)
```

Out[11]: <AxesSubplot:>

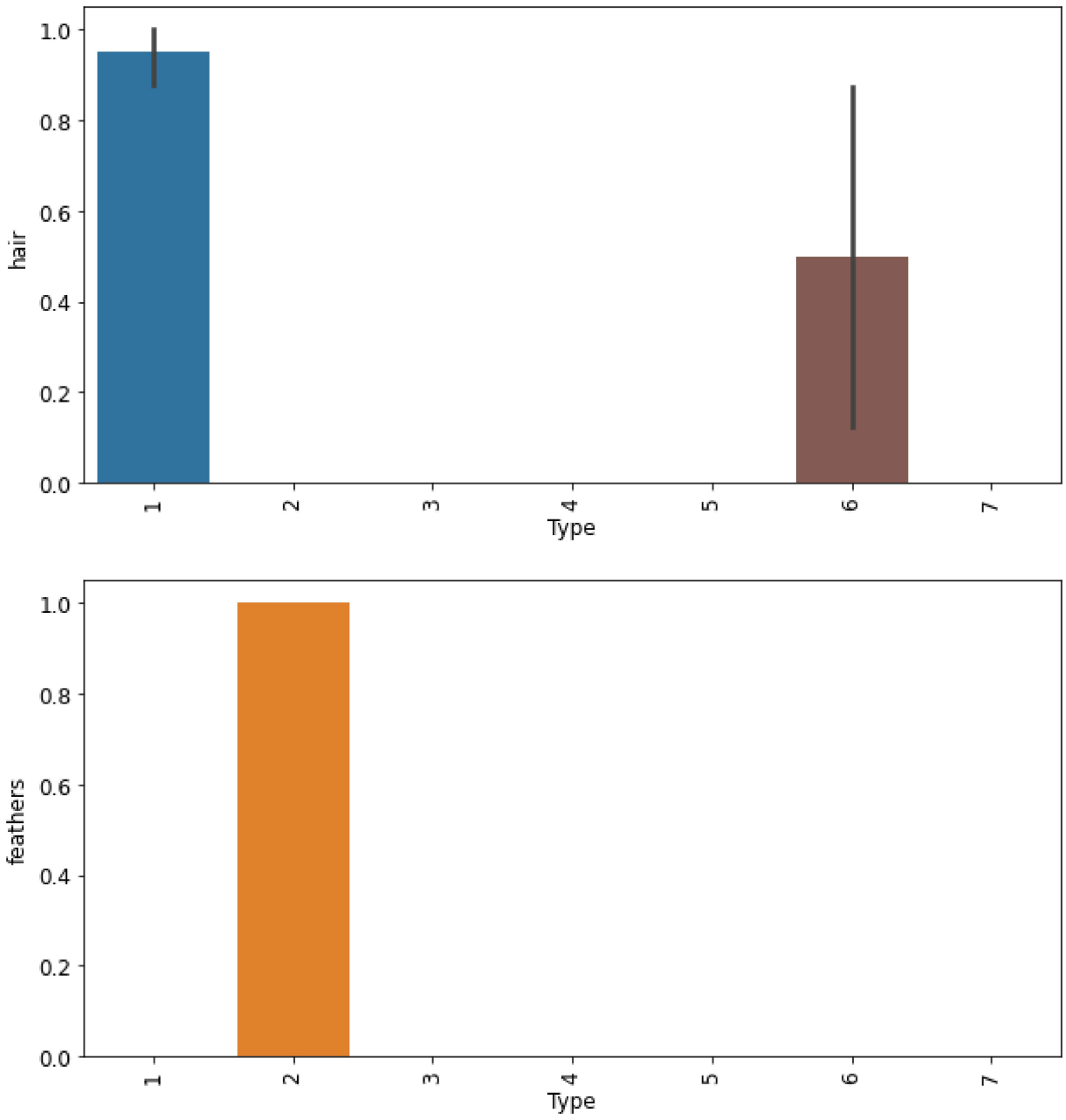


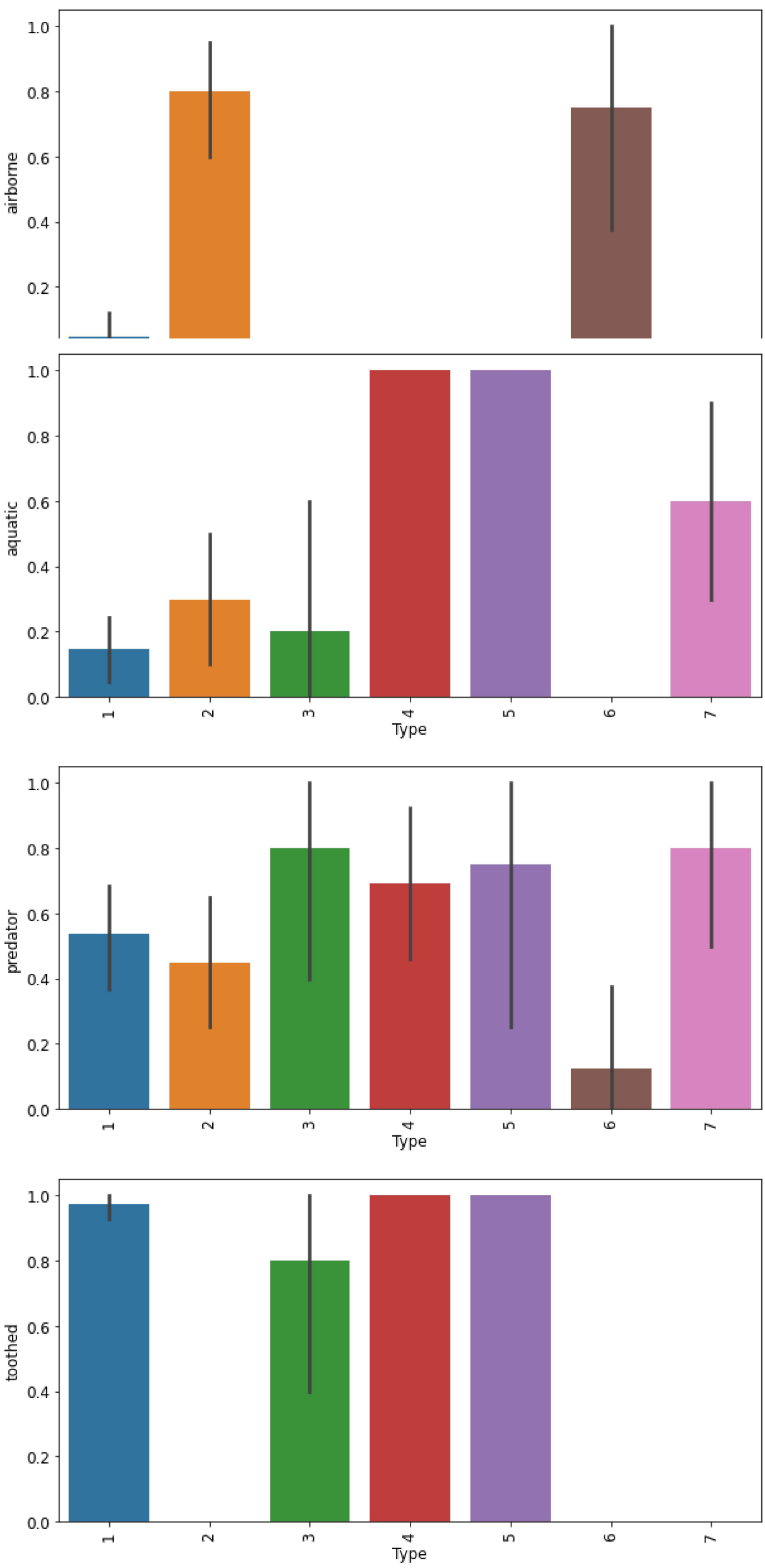
In [40]:
```
data2 = zoo.iloc[:,:-1]

correlations = data2.corrwith(zoo.type)
correlations = correlations[correlations!=1]
positive_correlations = correlations[correlations >0].sort_values(ascending = False)
negative_correlations =correlations[correlations<0].sort_values(ascending = False)

correlations.plot.bar(
        figsize = (10, 5),
        fontsize = 15,
        color = 'grey',
        rot = 45, grid = True)
plt.title('Correlation with zoo animal type \n',
horizontalalignment="center", fontstyle = "normal",
fontsize = "22", fontfamily = "sans-serif")
```

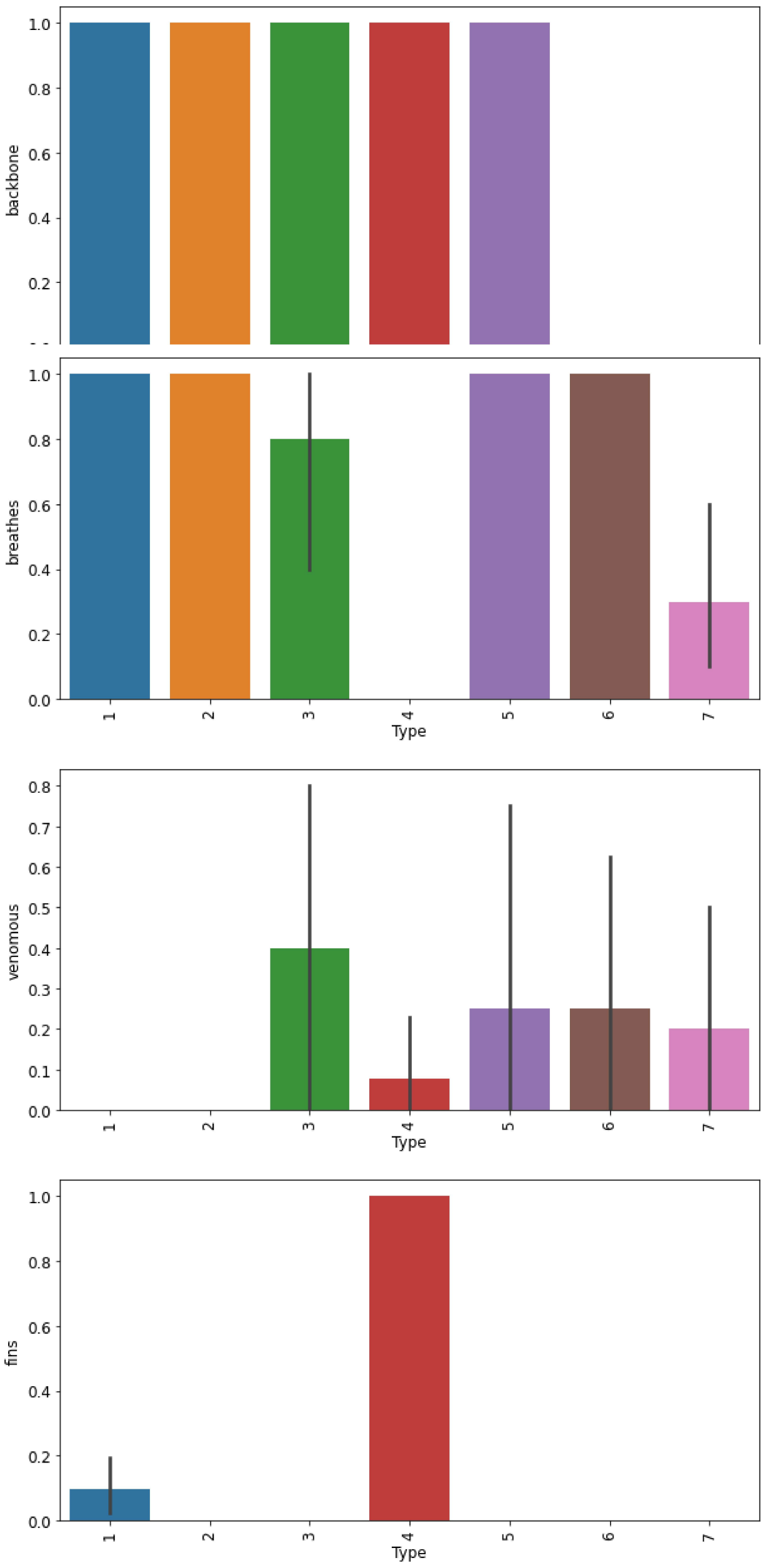
Out[40]: Text(0.5, 1.0, 'Correlation with zoo animal type \n')

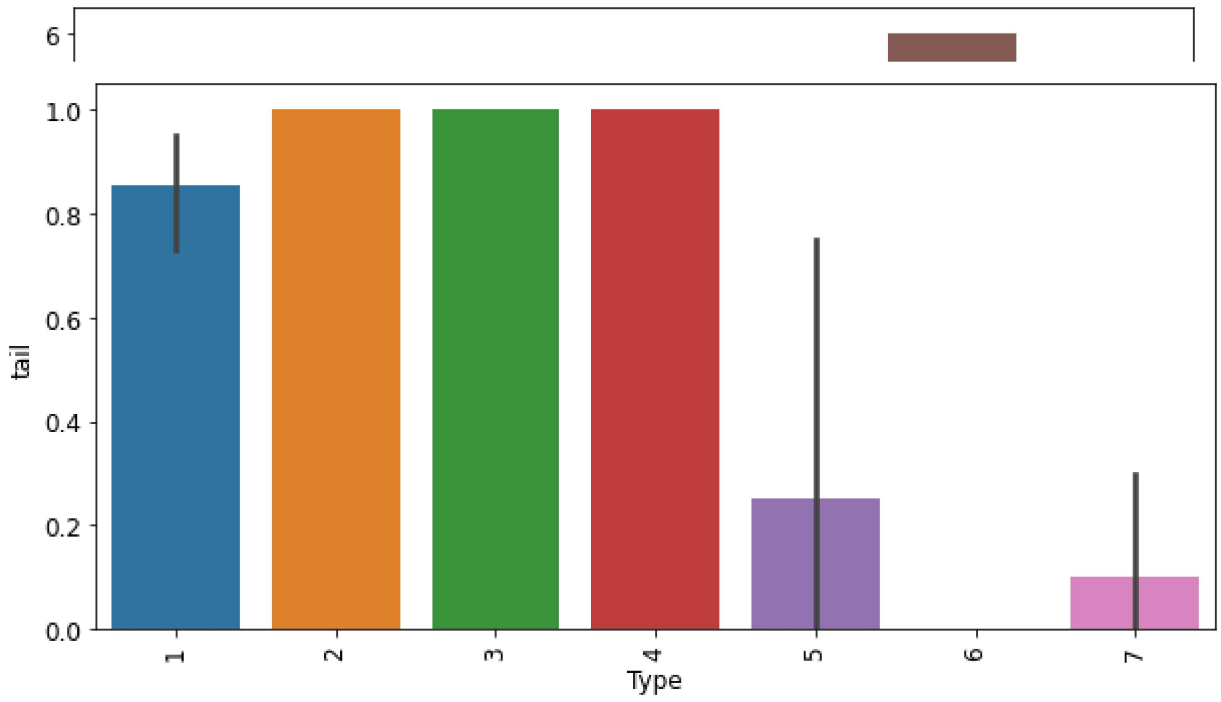


In [37]:
```
for i in zoo.columns[1:-1]:
    plt.figure(figsize=(10,5))
    sns.barplot(x = 'type', y= i,data = zoo)
    plt.xticks(rotation = 90, size = 12)
    plt.yticks(size = 12)
    plt.xlabel('Type',fontsize=12)
    plt.ylabel(i, fontsize=12)

    plt.show()
```

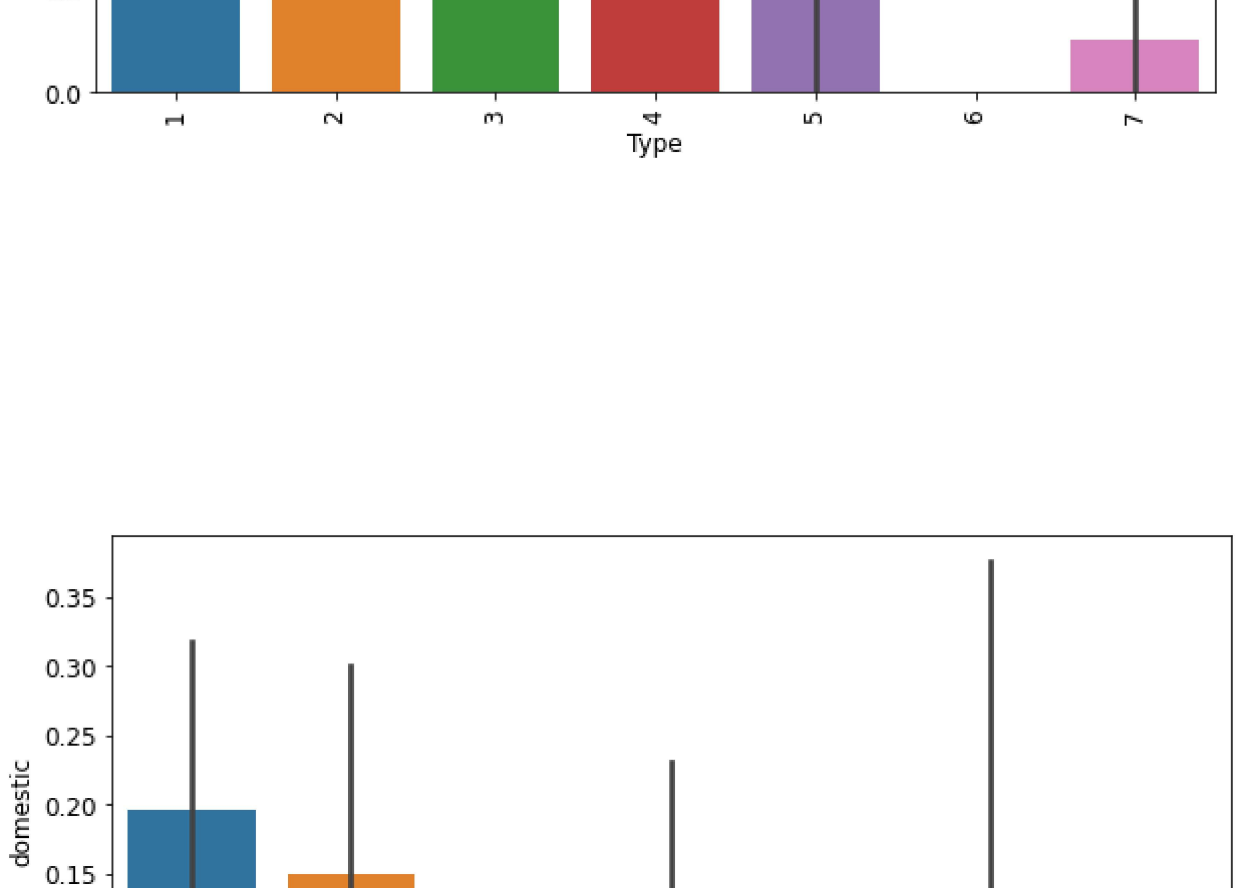



airborne
aquatic
predator
toothed
Type
1
2
3
4
5
6
7
0.0
0.2
0.4
0.6
0.8
1.0

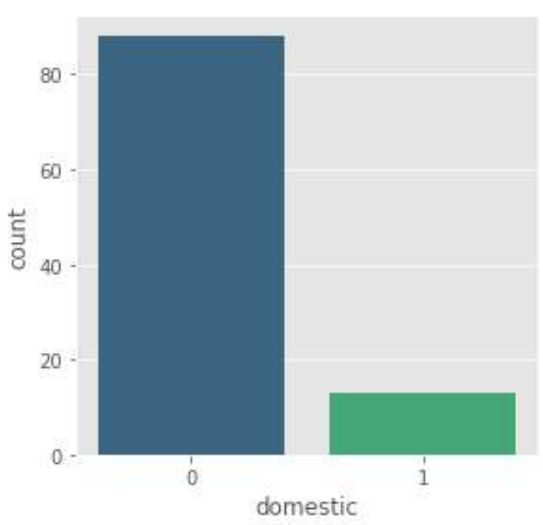

backbone
breathes
venomous
fins
Type
1
2
3
4
5
6
7

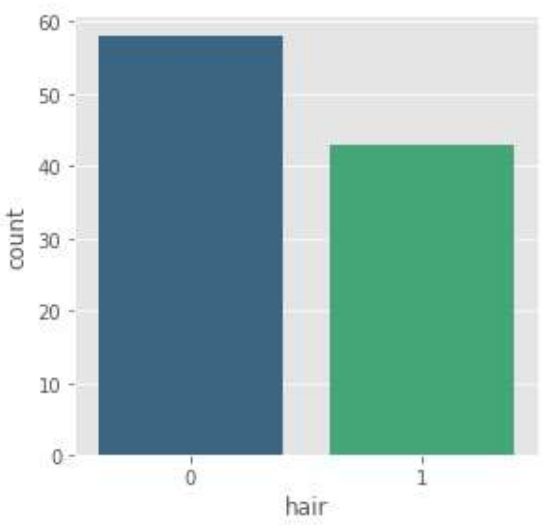

6
1.0
0.8
0.6
0.4
0.2
0.0
tail
1
2
3
4
5
6
7
Type

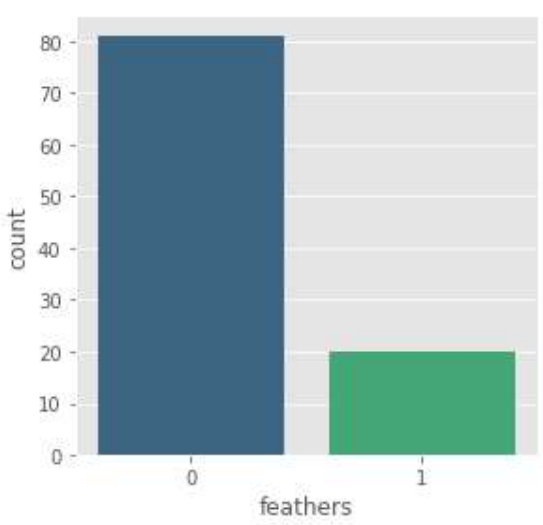
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
domestic
1
2
3
4
5
6
7
Type

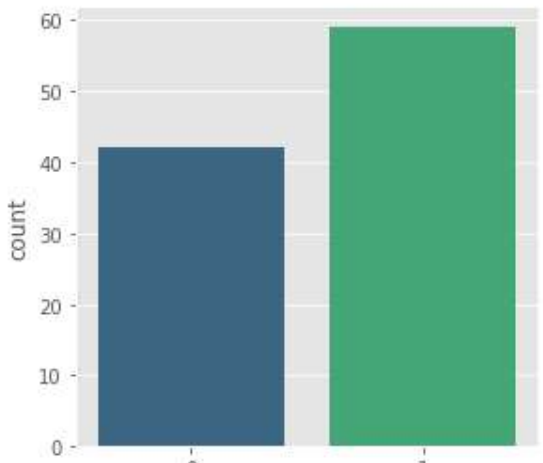

0.8
0.6
0.4
0.2
0.0
catsize
1
2
3
4
5
6
7
Type

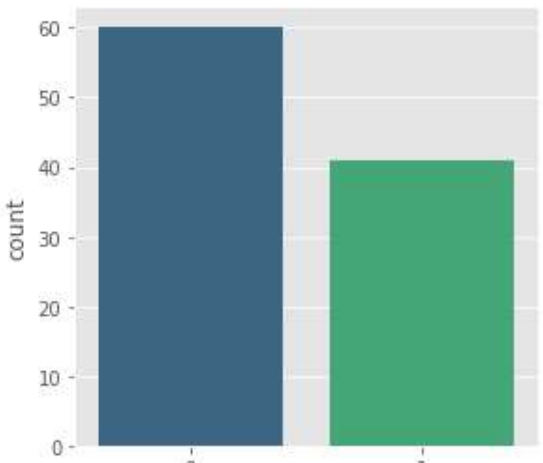
In [53]:
```
plt.style.use("ggplot")
fig, ax =plt.subplots(5,3, figsize=(15,25))

sns.countplot(zoo["domestic"], palette="viridis", ax=ax[0,0])
sns.countplot(zoo["hair"], palette="viridis", ax=ax[0,1])
sns.countplot(zoo["feathers"], palette="viridis", ax=ax[0,2])

sns.countplot(zoo["eggs"], palette="viridis", ax=ax[1,0])
sns.countplot(zoo["milk"], palette="viridis", ax=ax[1,1])
sns.countplot(zoo["airborne"], palette="viridis", ax=ax[1,2])

sns.countplot(zoo["aquatic"], palette="viridis", ax=ax[2,0])
sns.countplot(zoo["predator"], palette="viridis", ax=ax[2,1])
sns.countplot(zoo["toothed"], palette="viridis", ax=ax[2,2])

sns.countplot(zoo["backbone"], palette="viridis", ax=ax[3,0])
sns.countplot(zoo["breathes"], palette="viridis", ax=ax[3,1])
sns.countplot(zoo["venomous"], palette="viridis", ax=ax[3,2])

sns.countplot(zoo["fins"], palette="viridis", ax=ax[4,0])
sns.countplot(zoo["tail"], palette="viridis", ax=ax[4,1])
sns.countplot(zoo["catsize"], palette="viridis", ax=ax[4,2]);
```

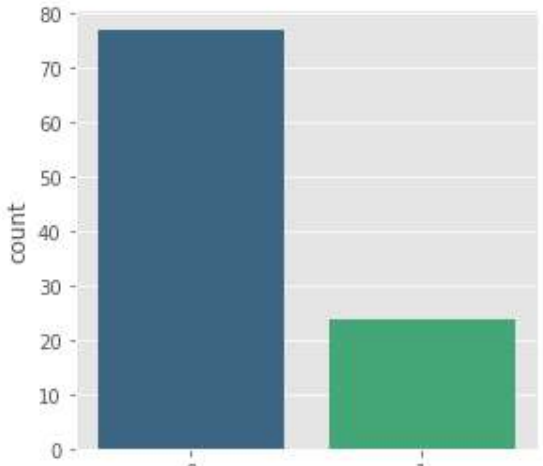

count
domestic

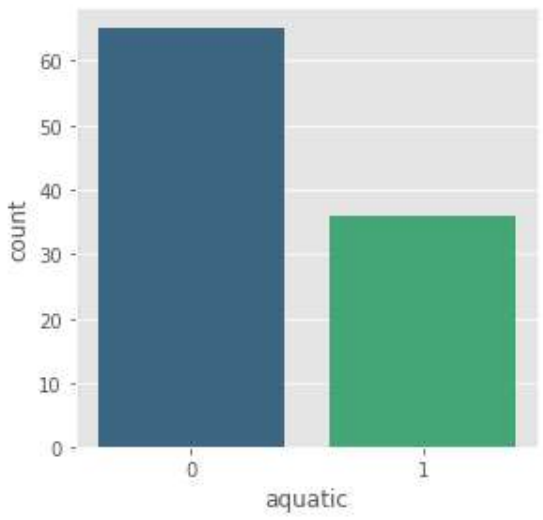

count
hair

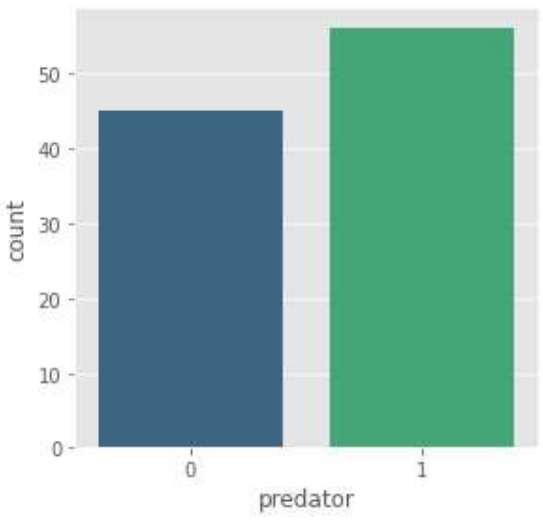

count
feathers

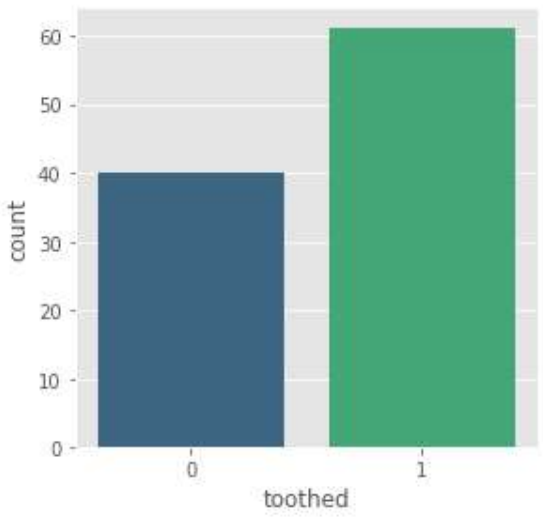

count
eggs

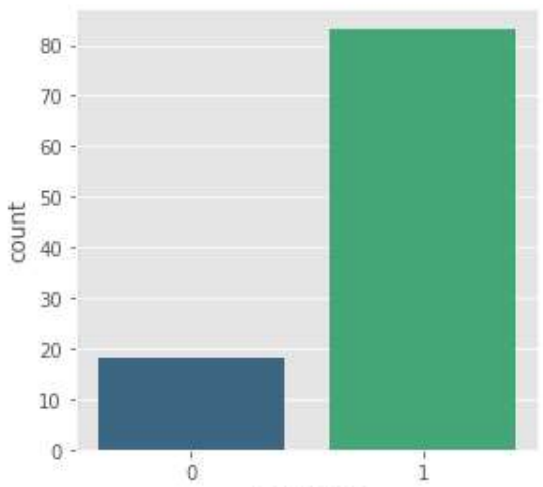

count
milk

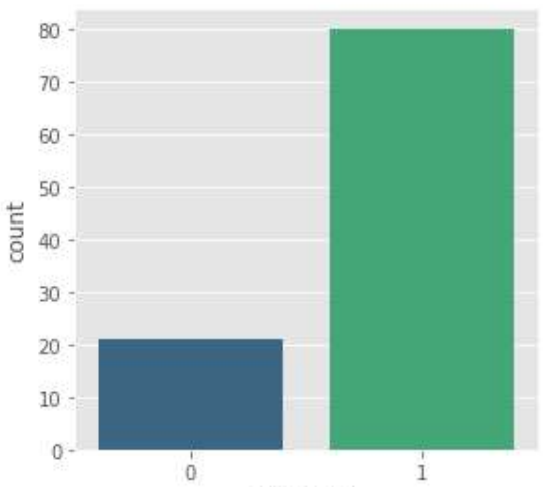

count
airborne

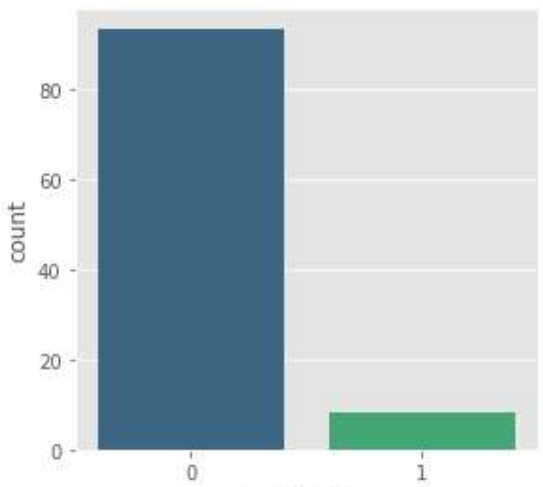

count
aquatic

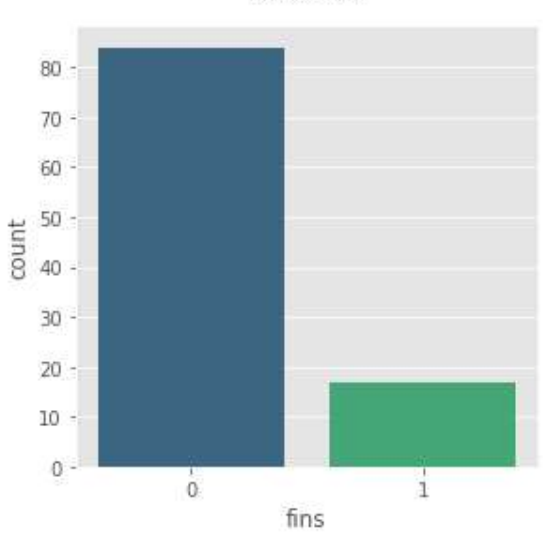

count
predator

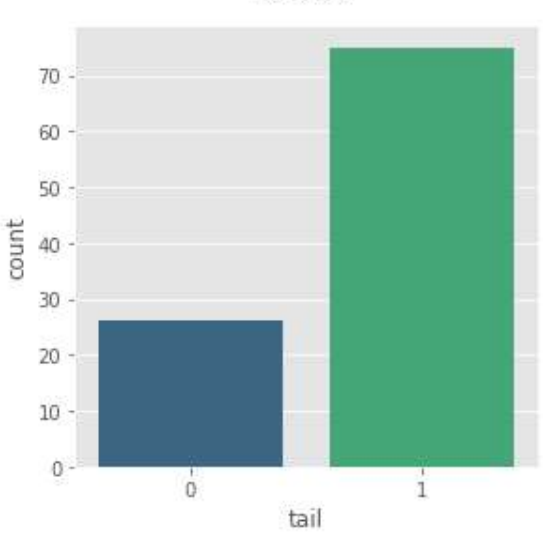

count
toothed

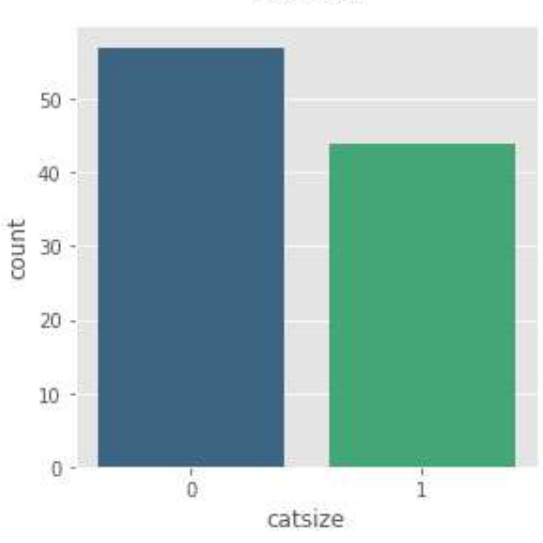

count
backbone


count
breathes


count
venomous


count
fins


count
tail


count
catsize

In [41]:
```
zoo_data = zoo.drop('animal name', axis = 1)
zoo_data.head()
```

Out[41]:

	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs	tail	domestic	catsize	ty
0	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	
1	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	
2	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	
3	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	
4	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	

In [42]:
```
X = zoo_data.drop('type', axis = 1)
y = zoo_data['type']
```

In [43]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,random_state = 0)
```

In [44]:
```
print('Shape of X_train: ', X_train.shape)
print('Shape of X_test: ', X_test.shape)
print('Shape of y_train: ', y_train.shape)
print('Shape of y_test: ', y_test.shape)
```

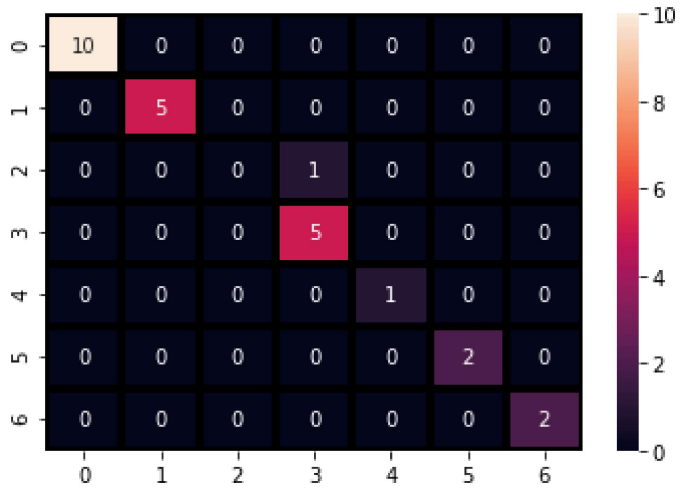
```
Shape of X_train:  (75, 16)
Shape of X_test:  (26, 16)
Shape of y_train:  (75,)
Shape of y_test:  (26,)
```

In [46]:
```
model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train, y_train)

pred = model.predict(X_test)
acc= accuracy_score(y_test,pred)
print("The accuracy is  {}".format(acc))
```

```
The accuracy is  0.9615384615384616
```

In [52]:
```
sns.heatmap(confusion_matrix(y_test, pred),annot=True,fmt = "d",linecolor="k",linewidths=3)
print('Classification Report ',classification_report(y_test,pred))
```

```
Classification Report                precision    recall  f1-score   support

           1       1.00      1.00      1.00        10
           2       1.00      1.00      1.00         5
           3       0.00      0.00      0.00         1
           4       0.83      1.00      0.91         5
           5       1.00      1.00      1.00         1
           6       1.00      1.00      1.00         2
           7       1.00      1.00      1.00         2

    accuracy                           0.96        26
   macro avg       0.83      0.86      0.84        26
weighted avg       0.93      0.96      0.94        26
```

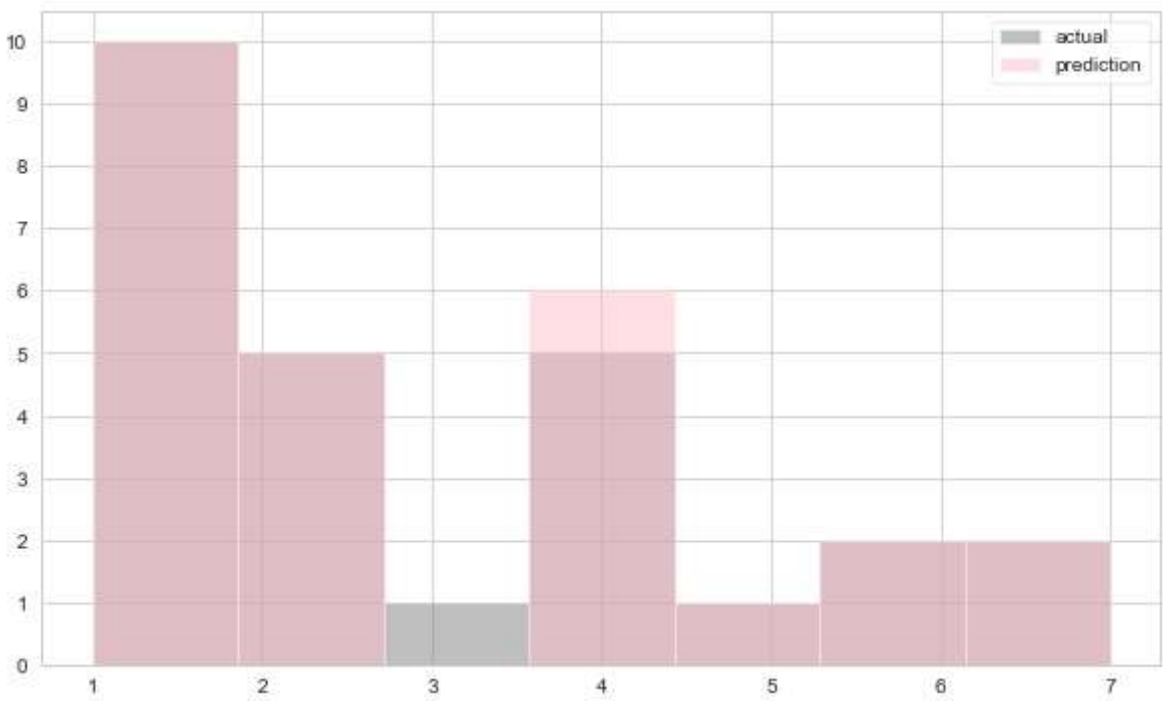


In [57]:
```
pred_df = pd.DataFrame({'Actual' : y_test, 'Predicted' : pred})
pred_df
```

Out[57]:

	Actual	Predicted
26	5	5
86	4	4
2	4	4
55	1	1
75	1	1
94	1	1
16	2	2
73	4	4
54	1	1
96	1	1
53	7	7
93	1	1
78	2	2
13	7	7
7	4	4
30	6	6
22	1	1
24	6	6
33	2	2
8	4	4
43	2	2
62	3	4
3	1	1
71	2	2
45	1	1
48	1	1

In [62]:
```
sns.set_style('whitegrid')

plt.rcParams['figure.figsize'] = (10, 6)
_, ax = plt.subplots()
ax.hist(y_test, color = 'grey', alpha = 0.5, label = 'actual', bins=7)
ax.hist(pred, color = 'pink', alpha = 0.5, label = 'prediction', bins=7)
ax.yaxis.set_ticks(np.arange(0,11))
ax.legend(loc = 'best')
plt.show()
```

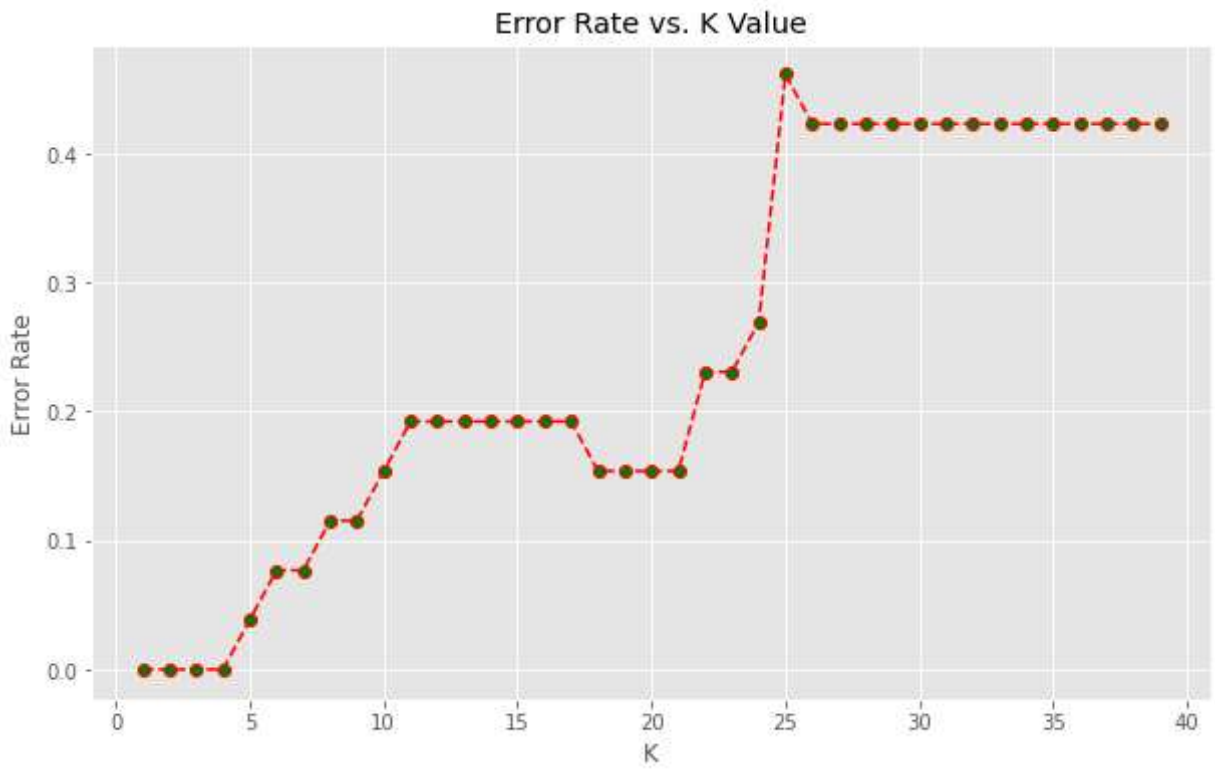


In [ ]:

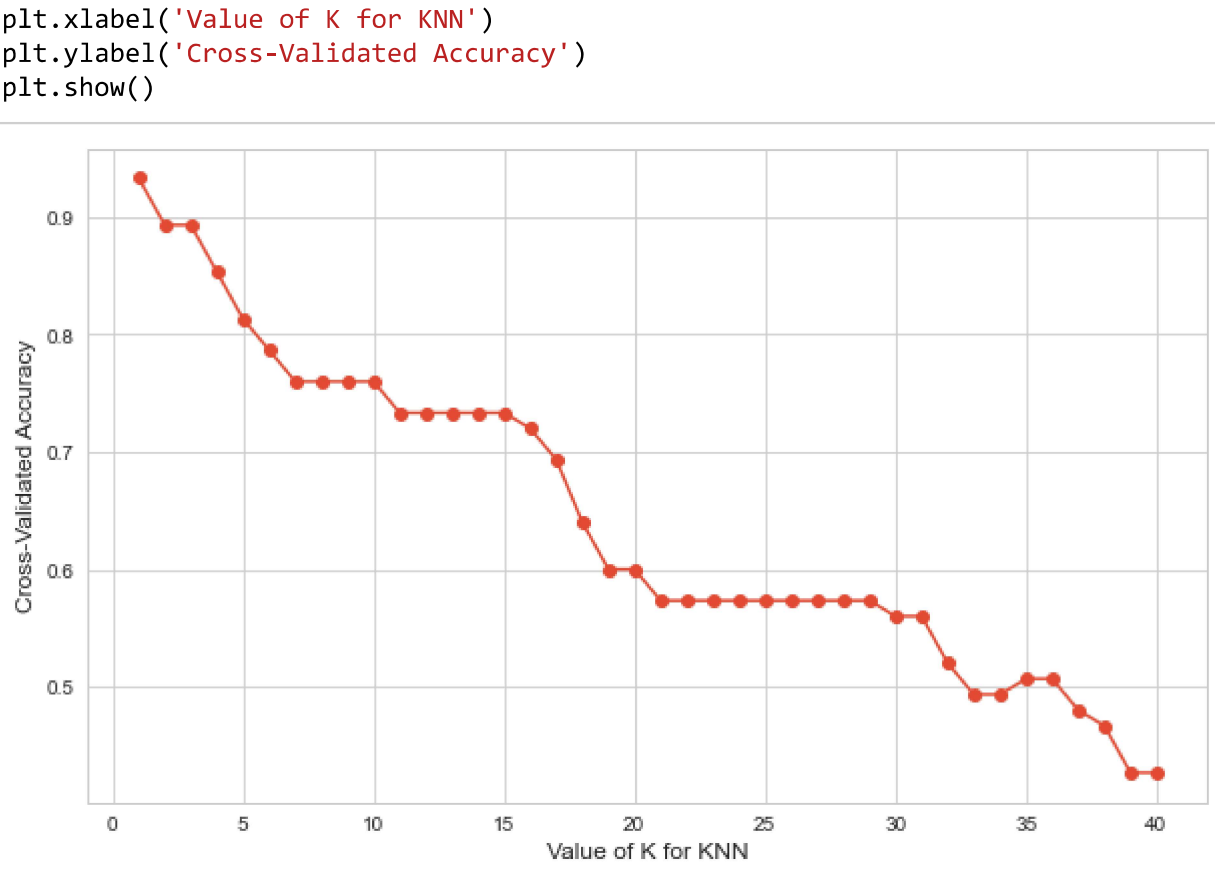
In [ ]:

In [55]:
```
err_rate=[]
for i in range(1,40):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    predict=knn.predict(X_test)
    err_rate.append(np.mean(predict!=y_test))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),err_rate,color='red', linestyle='dashed', marker='o',
         markerfacecolor='green', markersize=6)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[55]: Text(0, 0.5, 'Error Rate')



In [63]:
```
import matplotlib.pyplot as plt
%matplotlib inline

# choose k between 1 to 41
k_range = range(1, 41)
k_scores = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=5)
    k_scores.append(scores.mean())
# plot to see clearly
plt.figure(figsize=(10,6))
plt.plot(k_range, k_scores, marker ="o")
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
plt.show()
```



In [87]:
```
# parameters selection
kf = KFold(n_splits=15)
grid_params ={
    'n_neighbors':[1,2,3,4,5,6],
    'weights':['uniform','distance'],
    'metric' :['eclidean', 'manhattan']
}

gs = GridSearchCV(KNeighborsClassifier(), grid_params, verbose =10, cv=6,n_jobs=-1)
gs_results = gs.fit(X_train, y_train)
```

```
Fitting 6 folds for each of 24 candidates, totalling 144 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Batch computation too fast (0.0256s.) Setting batch_size=2.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Batch computation too fast (0.0830s.) Setting batch_size=4.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Batch computation too fast (0.1288s.) Setting batch_size=8.
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 100 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 120 tasks      | elapsed:    0.3s
[Parallel(n_jobs=-1)]: Done 144 out of 144 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 144 out of 144 | elapsed:    0.3s finished
```

In [88]:
```
gs_results.best_score_
```

Out[88]:
```
0.9476495726495727
```

In [90]:
```
gs_results.best_estimator_
```

Out[90]:
```
KNeighborsClassifier(metric='manhattan', n_neighbors=3)
```

In [91]:
```
model_final = gs_results.best_estimator_


pred_final = model_final.predict(X_test)
acc_final= accuracy_score(y_test,pred_final)
print("The accuracy is  {}".format(acc_final))
```

```
The accuracy is  1.0
```

In [92]:
```
sns.heatmap(confusion_matrix(y_test, pred_final),annot=True,fmt = "d",linecolor="k",linewidths=3)
print('Classification Report ',classification_report(y_test,pred_final))
```

```
Classification Report                precision    recall  f1-score   support

           1       1.00      1.00      1.00        10
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00         1
           4       1.00      1.00      1.00         5
           5       1.00      1.00      1.00         1
           6       1.00      1.00      1.00         2
           7       1.00      1.00      1.00         2

    accuracy                           1.00        26
   macro avg       1.00      1.00      1.00        26
weighted avg       1.00      1.00      1.00        26
```