In [1]:

```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt

import re


%matplotlib inline

import pandas_profiling # утилита, удобна для знакомства с данными
# с её помощью можно получить кучу описательных статистик по датасету

from wordcloud import WordCloud

import os

from PIL import Image

from sklearn.model_selection import train_test_split

# Кодирование категориальных переменных
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import category_encoders as ce

# Шкалирование переменных
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score, roc_curve

# Модели
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.model_selection import StratifiedKFold

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

import xgboost as xgb
import time
from sklearn.metrics import cohen_kappa_score, make_scorer
from imblearn.over_sampling import SMOTE

from sklearn.linear_model import SGDClassifier, LogisticRegressionCV
# SGD - стохастический градиентный спуск

import sklearn
import warnings
warnings.filterwarnings("ignore")
import nltk
from nltk.corpus import stopwords
import pymorphy2
from nltk.stem import WordNetLemmatizer
```

```
from nltk.corpus import wordnet
import string
from sklearn.feature_extraction.text import CountVectorizer
```

In [2]:

```
data = pd.read_csv("train/train.csv")

breeds = pd.read_csv('../petfinder/breed_labels.csv') # словарь пород
colors = pd.read_csv('../petfinder/color_labels.csv') # словарь окрасов шерсти
states = pd.read_csv('../petfinder/state_labels.csv') # словарь местоположения

data.shape
```

Out[2]:

```
(14993, 24)
```

In [3]:

```
missing = data.isnull().sum().to_frame(name = "count_of_missing_values")
missing["%"] = data.isnull().sum() / len(data)
missing
```

	count_of_missing_values	%
MaturitySize	0	0.000000
FurLength	0	0.000000
Vaccinated	0	0.000000
Dewormed	0	0.000000
Sterilized	0	0.000000
Health	0	0.000000
Quantity	0	0.000000
Fee	0	0.000000
State	0	0.000000
RescuerID	0	0.000000
VideoAmt	0	0.000000
Description	12	0.000800

In [4]:

```
data["Description"].fillna('', inplace=True)
```

In [5]:

```
data = data[["Description", "AdoptionSpeed"]]
```

In [6]:

```
data["Description"] = data["Description"].apply(lambda x: x.lower())
data["Description"][0]
```

Out[6]:

```
"nibble is a 3+ month old ball of cuteness. he is energetic and playful. i r
escued a couple of cats a few months ago but could not get them neutered in
time as the clinic was fully scheduled. the result was this little kitty. i
do not have enough space and funds to care for more cats in my household. lo
oking for responsible people to take over nibble's care."
```

In [7]:

```
stop = stopwords.words('english')
stemmer = nltk.stem.snowball.EnglishStemmer('english')

def data_Stemmer(text):
    return ' '.join([re.sub("\W", "", stemmer.stem(word)) for word in text.split(' ') if wo
```

In [8]:

```
def get_wordnet_pos(word):
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

def data_Lemmatizer(text):
    lemmatizer = WordNetLemmatizer()
    return " ".join([re.sub("\W", "", lemmatizer.lemmatize(w, get_wordnet_pos(w))) for w in
```

In [9]:

```
index = 0
print(f"Text = {data['Description'][index]} \n")
print(f"Stemming = {data_Stemmer(data['Description'][index])} \n")
print(f"Lemmatiz = {data_Lemmatizer(data['Description'][index])} \n")
```

```
Text = nibble is a 3+ month old ball of cuteness. he is energetic and playfu
l. i rescued a couple of cats a few months ago but could not get them neuter
ed in time as the clinic was fully scheduled. the result was this little kit
ty. i do not have enough space and funds to care for more cats in my househo
ld. looking for responsible people to take over nibble's care.

Stemming = nibbl 3 month old ball cuteness energet playful rescu coupl cat m
onth ago could get neuter time clinic fulli scheduled result littl kitty eno
ugh space fund care cat household look respons peopl take nibbl care

Lemmatiz = nibble 3 month old ball cuteness energetic playful rescue couple
cat month ago could get neuter time clinic fully schedule result little kitt
y enough space fund care cat household look responsible people take nibble s
care
```

In [10]:

```
t_start = time.time()
data_stem = data.copy()
data_stem["Description"] = data_stem["Description"].apply(lambda x: data_Stemmer(x))
data_stem["Description"][0]
t_end = time.time()
print(f"Time = {t_end - t_start} seconds")
```

```
Time = 13.74820852279663 seconds
```

In [11]:

```
t_start = time.time()
data_lem = data.copy()
data_lem["Description"] = data_lem["Description"].apply(lambda x: data_Lemmatizer(x))
data_lem["Description"][0]
t_end = time.time()
print(f"Time = {t_end - t_start} seconds")
```

```
Time = 641.3540880680084 seconds
```

In [12]:

```
X = data_stem["Description"]
y = data_stem["AdoptionSpeed"]
```

In [13]:

```
from sklearn.feature_extraction.text import TfidfTransformer, TfidfVectorizer
tf_vectorizer = TfidfVectorizer(stop_words=stop, ngram_range=(1, 2), min_df=0.001, norm='l2
```

In [14]:

```
X_train, X_test, y_train, y_test  = train_test_split(X, y, test_size=0.30, random_state=42)
```

In [15]:

```
X_train_tfidf = tf_vectorizer.fit_transform(X_train)
X_test_tfidf = tf_vectorizer.transform(X_test)
X_train_tfidf.shape, X_test_tfidf.shape
```

Out[15]:

```
((10495, 6015), (4498, 6015))
```

In [46]:

```
sklearn_vectorizer = CountVectorizer(stop_words='english', ngram_range=(1, 2), min_df=0.001
X_train_vect = sklearn_vectorizer.fit_transform(X_train)
X_test_vect = sklearn_vectorizer.transform(X_test)
X_train_vect.shape, X_test_vect.shape
```

Out[46]:

```
((10495, 5358), (4498, 5358))
```

In [47]:

```
X_l = data_lem["Description"]
y_l = data_lem["AdoptionSpeed"]

X_train_l, X_test_l, y_train_l, y_test_l  = train_test_split(X_l, y_l, test_size=0.30, rand

X_train_tfidf_l = tf_vectorizer.fit_transform(X_train_l)
X_test_tfidf_l = tf_vectorizer.transform(X_test_l)
X_train_tfidf_l.shape, X_test_tfidf_l.shape
```

Out[47]:

```
((10495, 6090), (4498, 6090))
```

In [48]:

```
X_train_vect_l = sklearn_vectorizer.fit_transform(X_train_l)
X_test_vect_l = sklearn_vectorizer.transform(X_test_l)
X_train_vect_l.shape, X_test_vect_l.shape
```

Out[48]:

```
((10495, 5282), (4498, 5282))
```

baseline

In [52]:

```
y_naive_pred = np.random.choice(
    [4., 2., 3., 1., 0.],
    len(y_test),
    p=y_train.value_counts(normalize=True).values) # p - вероятности
```

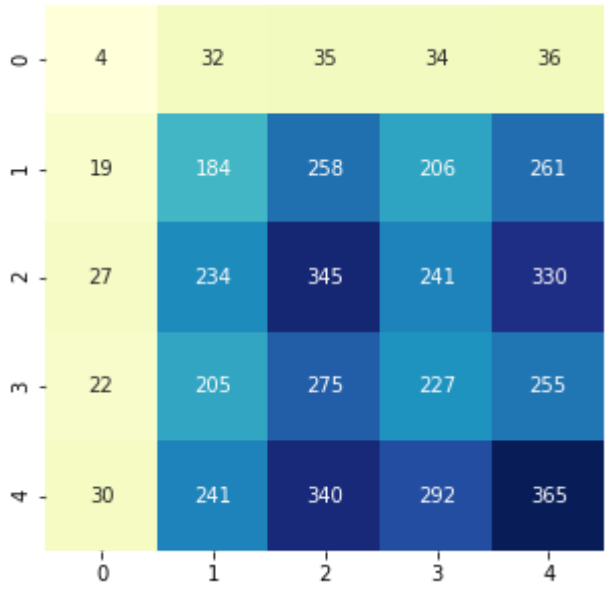
In [53]:

```
print(cohen_kappa_score(y_test, y_naive_pred, weights='quadratic'))
```

```
0.010101483137738843
```

In [54]:

```
cm = confusion_matrix(y_test, y_naive_pred)

conf_matrix = pd.DataFrame(data = cm, index=range(0, 5), columns=range(0, 5))
plt.figure(figsize = (5,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu", cbar=False);
# внизу предсказанные метки, слева - истинные метки
```



Используем для обучения линейные модели с использованием регуляризации

In [55]:

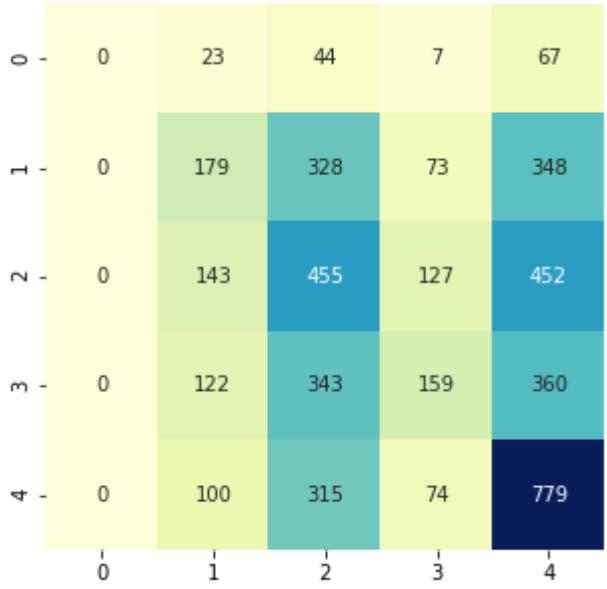
```
logit_sgd = SGDClassifier(
    loss='log', # так как в loss указано log, то получим классическую логистическую регресс
    # с методом обучения стохастического градиентного спуска.
    shuffle=True, # для перемешивания выборки (перемешивает - обучается, перемешивает - обу
    n_iter_no_change=10, # критерий остановки - число итераций без изменения функции потерь
    # если последовательно в течение 10 итераций не было улучшения score, то перестает учит
    max_iter=1000, # максимальное число итераций
    penalty='l1', # l1 - регуляризация
    random_state=42
)
```

SGDClassifier позволяет получить интерпретируемые оценки. Можно посмотреть наличие каких слов или словосочетаний приводит к той или иной скорости принятия питомца в семью

In [56]:

```
logit_sgd.fit(X_train_tfidf, y_train)

print(cohen_kappa_score(y_train, logit_sgd.predict(X_train_tfidf), weights='quadratic'))
print(cohen_kappa_score(y_test, logit_sgd.predict(X_test_tfidf), weights='quadratic'))

cm = confusion_matrix(y_test, logit_sgd.predict(X_test_tfidf))

conf_matrix = pd.DataFrame(data = cm, index=range(0, 5), columns=range(0, 5))
plt.figure(figsize = (5,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu", cbar=False);
# внизу предсказанные метки, слева - истинные метки
```

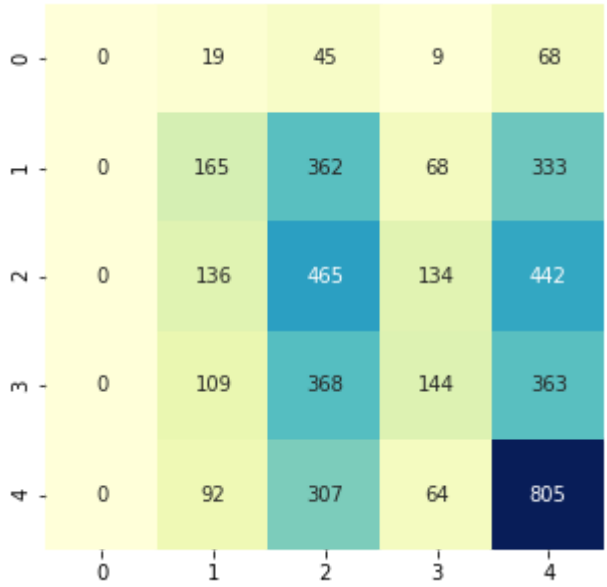
```
0.28765711913571734
0.15884389774857655
```



In [117]:

```
logit_sgd.fit(X_train_tfidf_l, y_train_l)

print(cohen_kappa_score(y_train_l, logit_sgd.predict(X_train_tfidf_l), weights='quadratic'))
print(cohen_kappa_score(y_test_l, logit_sgd.predict(X_test_tfidf_l), weights='quadratic'))

cm = confusion_matrix(y_test_l, logit_sgd.predict(X_test_tfidf_l))

conf_matrix = pd.DataFrame(data = cm, index=range(0, 5), columns=range(0, 5))
plt.figure(figsize = (5,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu", cbar=False);
```

```
0.27392100933355734
0.1744549420826853
```



In [32]:

```
# функция для подбора наилучших параметров
def grid_search_cv(model, param_grid, x_train, y_train):
    kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=7)
    kappa_scorer = make_scorer(cohen_kappa_score, weights = 'quadratic')
    grid_search = GridSearchCV(model, param_grid, cv=kfold, scoring=kappa_scorer, verbose=4
    t_start = time.time()
    grid_search.fit(x_train, y_train)
    t_end = time.time()
    print('model {} best score is {}'.format(model.__class__.__name__, grid_search.best_sco
    print('time for training is {} seconds'.format(t_end - t_start))
    print(f"Best params = {grid_search.best_estimator_}")
    return grid_search.best_estimator_
```

In [31]:

```
param_grid = {
    'loss': ['log'],
    'penalty': ['l1','l2'],
    'random_state': [42],
    'learning_rate': ['constant', 'optimal', 'invscaling', 'adaptive'],
    'n_iter_no_change': [5, 10, 15],
    'class_weight': ['balanced', None],
    'eta0': [0.01, 0.1, 0.05]
}
```

In [58]:

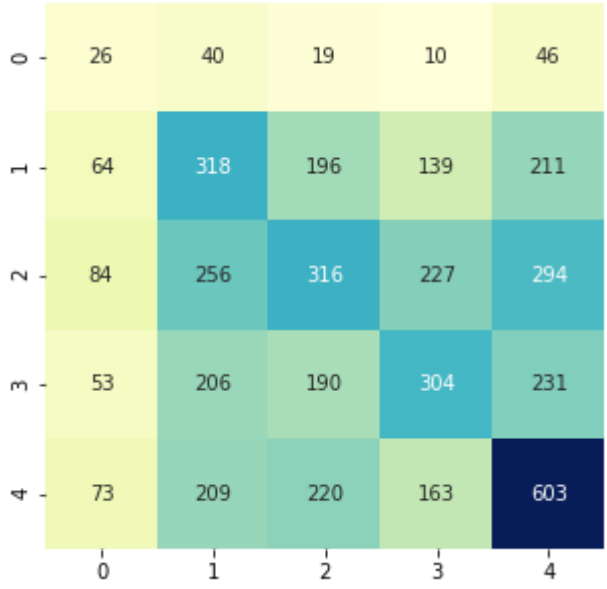
```
sgd = grid_search_cv(SGDClassifier(), param_grid, X_train_tfidf, y_train)
```

```
Fitting 10 folds for each of 144 candidates, totalling 1440 fits
[CV 1/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   0.9
s
[CV 2/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   0.8
s
[CV 3/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   0.6
s
[CV 4/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   0.7
s
[CV 5/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   0.9
s
[CV 6/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   0.8
s
```

In [17]:

```
t_start = time.time()
sgd = SGDClassifier(class_weight='balanced', eta0=0.1, learning_rate='constant',
              loss='log', random_state=42)
sgd.fit(X_train_tfidf, y_train)
t_end = time.time()

print(cohen_kappa_score(y_test, sgd.predict(X_test_tfidf), weights='quadratic'))
print(f"Time = {t_end - t_start} seconds")

cm = confusion_matrix(y_test, sgd.predict(X_test_tfidf))

conf_matrix = pd.DataFrame(data = cm, index=range(0, 5), columns=range(0, 5))
plt.figure(figsize = (5,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu", cbar=False);
```

```
0.18808613133170438
Time = 0.4965803623199463 seconds
```



In [120]:

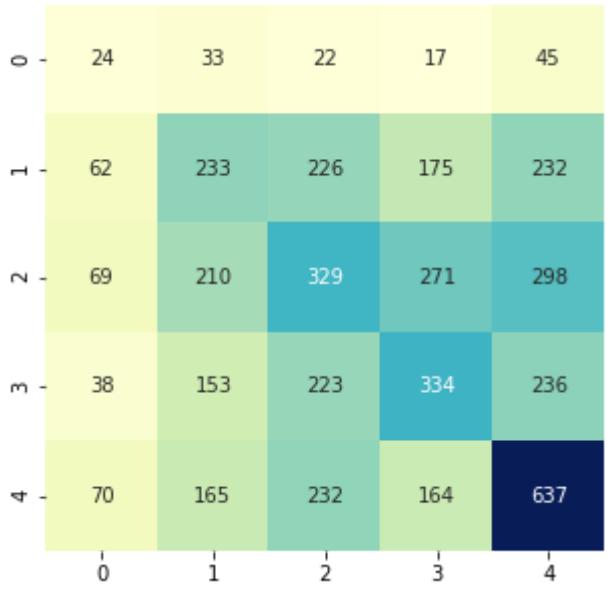
```
sgd = grid_search_cv(SGDClassifier(), param_grid, X_train_tfidf_l, y_train_l)
```

```
Fitting 10 folds for each of 144 candidates, totalling 1440 fits
[CV 1/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   1.0
s
[CV 2/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   0.9
s
[CV 3/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   0.8
s
[CV 4/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   0.9
s
[CV 5/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   0.8
s
[CV 6/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   0.8
s
```

In [121]:

```
t_start = time.time()
sgd = SGDClassifier(class_weight='balanced', eta0=0.01, loss='log', random_state=42)
sgd.fit(X_train_tfidf_l, y_train_l)
t_end = time.time()

print(cohen_kappa_score(y_test_l, sgd.predict(X_test_tfidf_l), weights='quadratic'))
print(f"Time = {t_end - t_start} seconds")

cm = confusion_matrix(y_test_l, sgd.predict(X_test_tfidf_l))

conf_matrix = pd.DataFrame(data = cm, index=range(0, 5), columns=range(0, 5))
plt.figure(figsize = (5,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu", cbar=False);
```

```
0.17813713494979488
Time = 0.2922213077545166 seconds
```



In [49]:

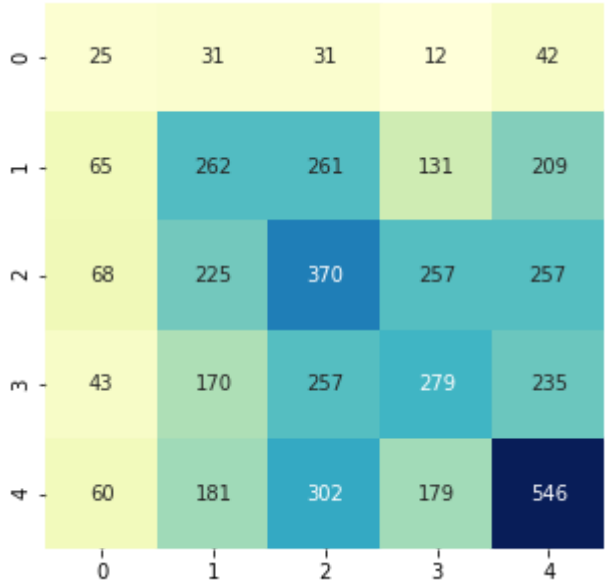
```
sgd = grid_search_cv(SGDClassifier(), param_grid, X_train_vect, y_train)
```

```
Fitting 10 folds for each of 144 candidates, totalling 1440 fits
[CV 1/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   1.7
s
[CV 2/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   1.6
s
[CV 3/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   2.0
s
[CV 4/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   1.8
s
[CV 5/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   1.8
s
[CV 6/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   1.7
s
```

In [51]:

```
t_start = time.time()
sgd = SGDClassifier(class_weight='balanced', eta0=0.01, learning_rate='constant',
              loss='log', penalty='l1', random_state=42)
sgd.fit(X_train_vect, y_train)
t_end = time.time()

print(cohen_kappa_score(y_test, sgd.predict(X_test_vect), weights='quadratic'))
print(f"Time = {t_end - t_start} seconds")

cm = confusion_matrix(y_test, sgd.predict(X_test_vect))

conf_matrix = pd.DataFrame(data = cm, index=range(0, 5), columns=range(0, 5))
plt.figure(figsize = (5,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu", cbar=False);
```

```
0.17881245672899904
Time = 2.3706562519073486 seconds
```



In [50]:

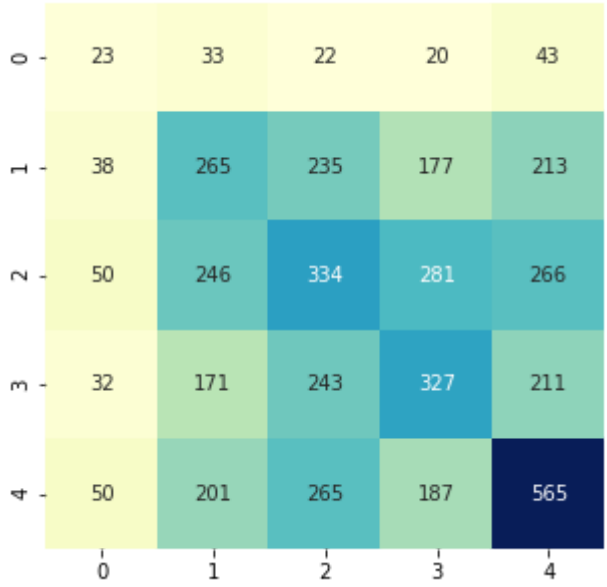
```
sgd = grid_search_cv(SGDClassifier(), param_grid, X_train_vect_l, y_train_l)
```

```
Fitting 10 folds for each of 144 candidates, totalling 1440 fits
[CV 1/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   1.7
s
[CV 2/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   1.6
s
[CV 3/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   1.9
s
[CV 4/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   2.0
s
[CV 5/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   1.7
s
[CV 6/10] END class_weight=balanced, eta0=0.01, learning_rate=constant, lo
ss=log, n_iter_no_change=5, penalty=l1, random_state=42; total time=   1.6
s
```

In [52]:

```
t_start = time.time()
sgd = SGDClassifier(class_weight='balanced', eta0=0.01, learning_rate='adaptive',
              loss='log', n_iter_no_change=15, random_state=42)
sgd.fit(X_train_vect_l, y_train_l)
t_end = time.time()

print(cohen_kappa_score(y_test_l, sgd.predict(X_test_vect_l), weights='quadratic'))
print(f"Time = {t_end - t_start} seconds")

cm = confusion_matrix(y_test_l, sgd.predict(X_test_vect_l))

conf_matrix = pd.DataFrame(data = cm, index=range(0, 5), columns=range(0, 5))
plt.figure(figsize = (5,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu", cbar=False);
```

```
0.1615664793128454
Time = 3.1864776611328125 seconds
```



In [32]:

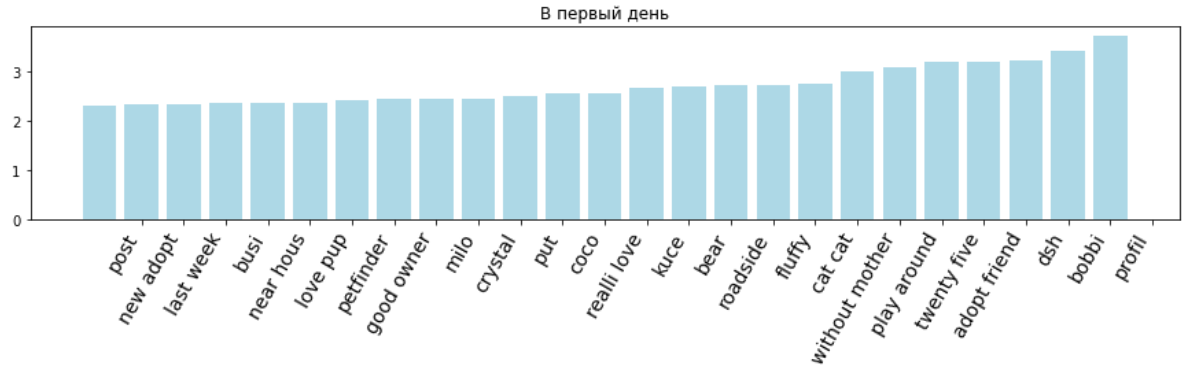
```
lst_feat = tf_vectorizer.get_feature_names()
lst_feat[lst_feat.index('25')] = 'twenty five'
```

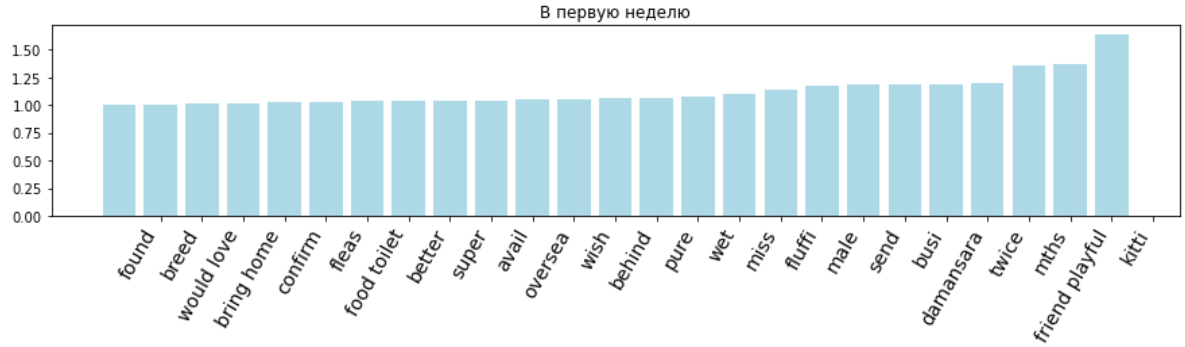
In [18]:

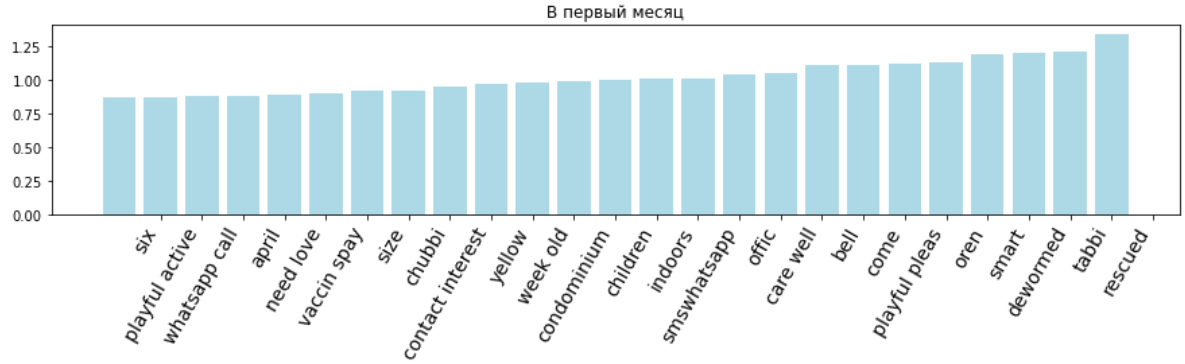
```
def visualize_coefficients(coef, feature_names, title, plot_num, n_top_features=25):
    interesting_coefficients = np.argsort(coef)[-n_top_features:]

    plt.figure(figsize=(15, 15))
    plt.subplot(510+plot_num+1)
    colors = ["lightblue" for c in coef[interesting_coefficients]]
    plt.bar(np.arange(n_top_features), coef[interesting_coefficients], color=colors)
    feature_names = np.array(feature_names)
    plt.xticks(np.arange(1, 1 + n_top_features),
               feature_names[interesting_coefficients], rotation=60, ha="right", fontsize=1
    plt.title(title)
```

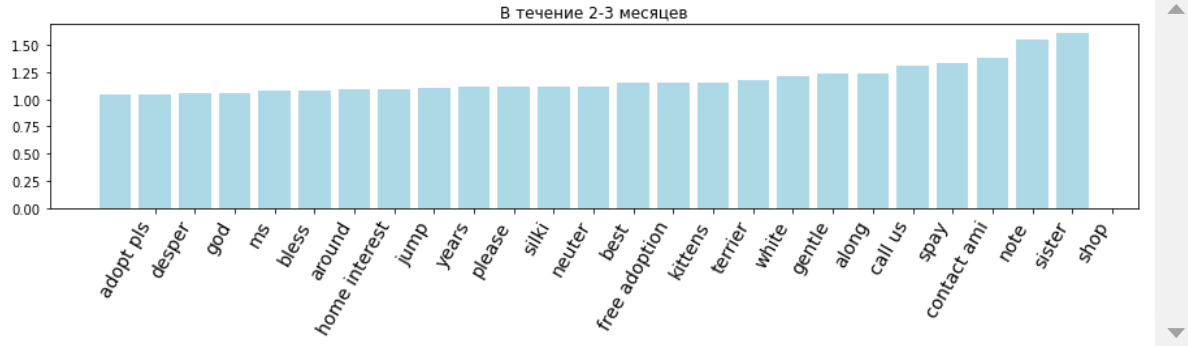
In [35]:

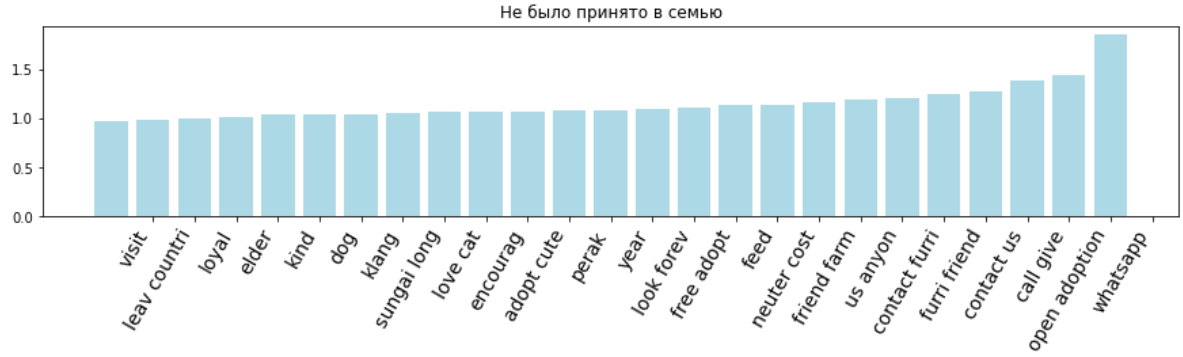
```
titles = ["В первый день", "В первую неделю", 'В первый месяц', 'В течение 2-3 месяцев', 'Н
for i in range(5):
    visualize_coefficients(sgd.coef_[i,:], lst_feat,
                           title=titles[i], plot_num=i)
```





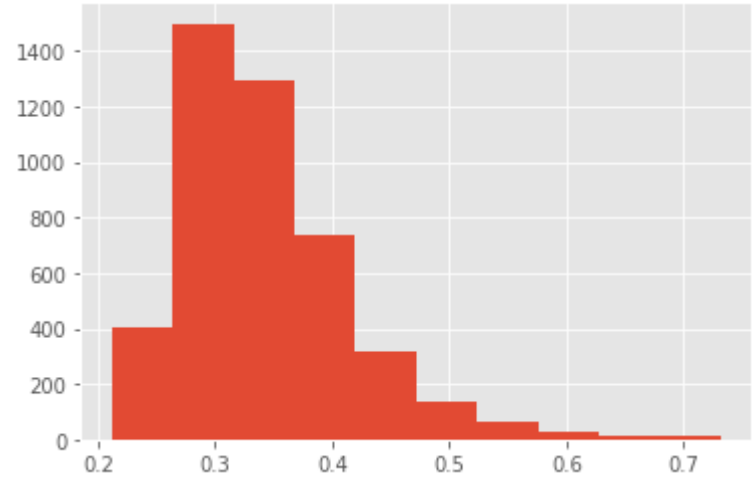






In [38]:

```
plt.style.use("ggplot")
plt.hist(np.max(sgd.predict_proba(X_test_tfidf), axis=1), bins=10);
```



In [ ]: