Pet Adoption Prediction Data Feature Engineering and Modelling

In [1]:
```
#libraries
import numpy as np
import pandas as pd
import os
import json
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
import lightgbm as lgb
import xgboost as xgb
import time
import datetime
from PIL import Image
from wordcloud import WordCloud
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold, KFold
from sklearn.metrics import mean_squared_error, roc_auc_score
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
import gc
from catboost import CatBoostClassifier
from tqdm import tqdm_notebook
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import random
import warnings
warnings.filterwarnings("ignore")
from functools import partial
pd.set_option('max_colwidth', 500)
pd.set_option('max_columns', 500)
pd.set_option('max_rows', 100)
import os
import scipy as sp
from math import sqrt
from collections import Counter
from sklearn.metrics import confusion_matrix as sk_cmatrix

from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import TweetTokenizer
from sklearn.ensemble import RandomForestClassifier
import langdetect
import eli5
from IPython.display import display

from sklearn.metrics import cohen_kappa_score
def kappa(y_true, y_pred):
    return cohen_kappa_score(y_true, y_pred, weights='quadratic')
```

In [2]:
```
breeds = pd.read_csv('../input/breed_labels.csv')
colors = pd.read_csv('../input/color_labels.csv')
states = pd.read_csv('../input/state_labels.csv')

train = pd.read_csv('../input/train/train.csv')
test = pd.read_csv('../input/test/test.csv')

train['dataset_type'] = 'train'
test['dataset_type'] = 'test'
all_data = pd.concat([train, test])
```

Data overview

In [3]:
```
print(os.listdir("../input"))
```

```
['state_labels.csv', 'train_metadata', 'color_labels.csv', 'StateLabels.csv', 'train_
images', 'test_metadata', 'PetFinder-BreedLabels.csv', 'BreedLabels.csv', 'train_sent
iment', 'ColorLabels.csv', 'test_sentiment', 'PetFinder-ColorLabels.csv', 'PetFinder-
StateLabels.csv', 'test', 'train', 'breed_labels.csv', 'test_images']
```

In [4]:
```
train.drop('Description', axis=1).head()
```

Out[4]:

	Type	Name	Age	Breed1	Breed2	Gender	Color1	Color2	Color3	MaturitySize	FurLength	Va
0	2	Nibble	3	299	0	1	1	7	0	1	1	
1	2	No Name Yet	1	265	0	1	1	2	0	2	2	
2	1	Brisco	1	307	0	1	2	7	0	2	2	
3	1	Miko	4	307	0	2	1	2	0	2	1	
4	1	Hunter	1	307	0	1	1	0	0	2	1	

In [5]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14993 entries, 0 to 14992
Data columns (total 25 columns):
Type             14993 non-null int64
Name             13736 non-null object
Age              14993 non-null int64
Breed1           14993 non-null int64
Breed2           14993 non-null int64
Gender           14993 non-null int64
Color1           14993 non-null int64
Color2           14993 non-null int64
Color3           14993 non-null int64
MaturitySize     14993 non-null int64
FurLength        14993 non-null int64
Vaccinated       14993 non-null int64
Dewormed         14993 non-null int64
Sterilized       14993 non-null int64
Health           14993 non-null int64
Quantity         14993 non-null int64
Fee              14993 non-null int64
State            14993 non-null int64
RescuerID        14993 non-null object
VideoAmt         14993 non-null int64
Description      14981 non-null object
PetID            14993 non-null object
PhotoAmt         14993 non-null float64
AdoptionSpeed    14993 non-null int64
dataset_type     14993 non-null object
dtypes: float64(1), int64(19), object(5)
memory usage: 2.9+ MB
```

We have almost 15,000 dogs and cats in the dataset. The main dataset contains all important information about pets: age, breed, color, various characteristics, and other details. Descriptions were analyzed using Google's Natural Language API, providing sentiments and entities. I believe we could perform a similar analysis ourselves. There are photos of some pets available. Certain meta-information was extracted from images and can be utilized. Separate files containing labels for breeds, colors, and states are available. Let's begin with the main dataset.

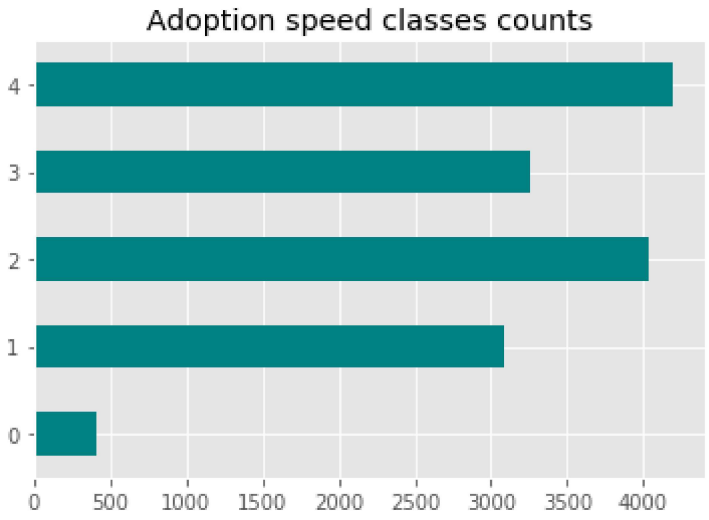
Additionally, I've created a comprehensive dataset by combining both train and test data purely for more convenient visualization. The "dataset_type" column indicates the dataset to which the data belongs.

Main data exploration

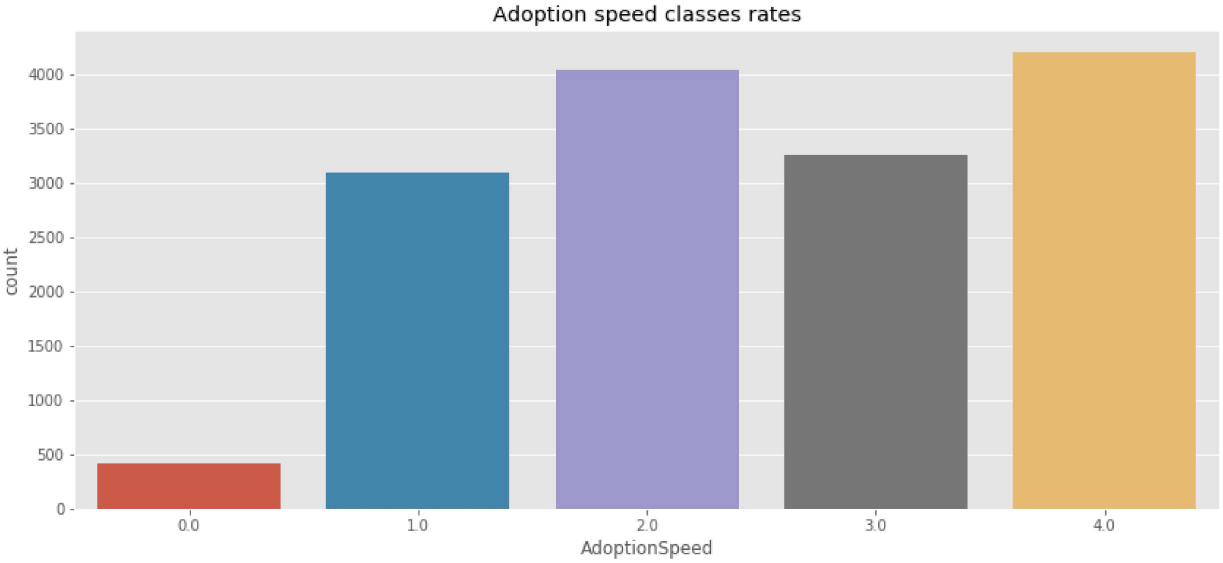
Target: Adoption speed

0 - Pet was adopted on the same day as it was listed. 1 - Pet was adopted between 1 and 7 days (1st week) after being listed. 2 - Pet was adopted between 8 and 30 days (1st month) after being listed. 3 - Pet was adopted between 31 and 90 days (2nd & 3rd month) after being listed. 4 - No adoption within the first 100 days of being listed. (There are no pets in this dataset that were not adopted after 90 days).

In [6]:
```
train['AdoptionSpeed'].value_counts().sort_index().plot('barh', color='teal');
plt.title('Adoption speed classes counts');
```



In [7]:
```
plt.figure(figsize=(14, 6));
g = sns.countplot(x='AdoptionSpeed', data=all_data.loc[all_data['dataset_type'] == 'tr
plt.title('Adoption speed classes rates');
ax=g.axes
```



In [8]:
```
#Axes
ax
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5c2323b400>
```

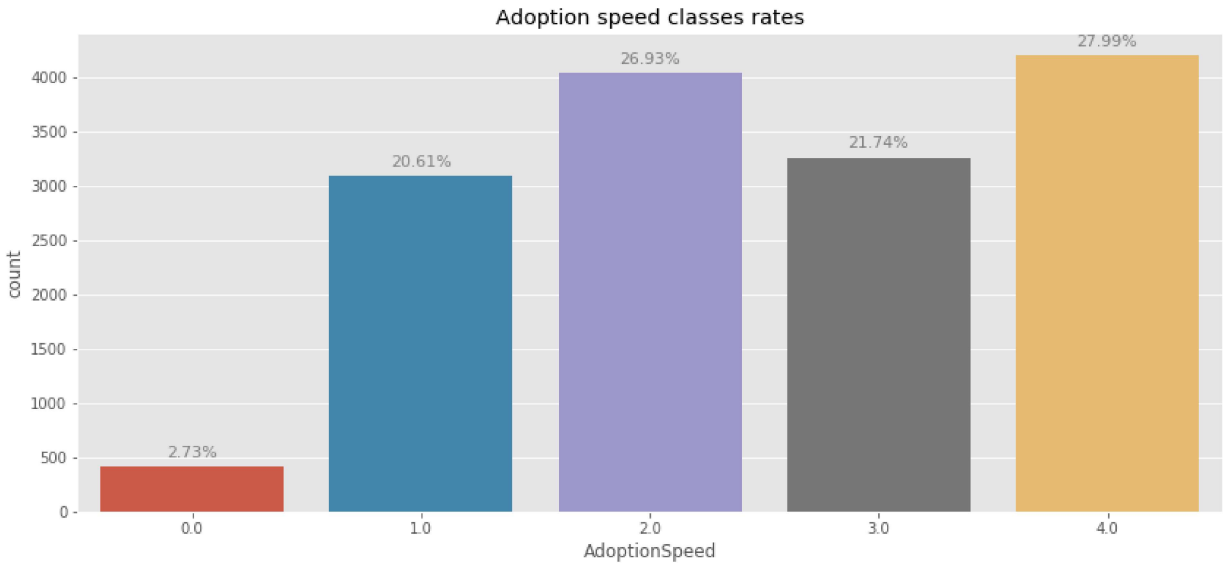
In [9]:
```
# patches
ax.patches
```

Out[9]:
```
[<matplotlib.patches.Rectangle at 0x7f5c229fd518>,
 <matplotlib.patches.Rectangle at 0x7f5c229fd7f0>,
 <matplotlib.patches.Rectangle at 0x7f5c229fdb38>,
 <matplotlib.patches.Rectangle at 0x7f5c229fde80>,
 <matplotlib.patches.Rectangle at 0x7f5c22986208>]
```

In [10]:
```
# example of info in patches
ax.patches[0].get_x()
```

Out[10]:
```
-0.4
```

In [11]:
```
plt.figure(figsize=(14, 6));
g = sns.countplot(x='AdoptionSpeed', data=all_data.loc[all_data['dataset_type'] == 'tr
plt.title('Adoption speed classes rates');
ax=g.axes
for p in ax.patches:
     ax.annotate(f"{p.get_height() * 100 / train.shape[0]:.2f}%", (p.get_x() + p.get_w
         ha='center', va='center', fontsize=11, color='gray', rotation=0, xytext=(0, 1
         textcoords='offset points')
```



We observed that some pets are adopted immediately, though these cases are rare, perhaps driven by a general interest in adopting any available pet or sheer luck in catching a potential adopter's eye.

Regrettably, a significant number of pets remain unadopted, which is indeed disheartening. I'm hopeful that our models and analyses will contribute to finding suitable homes for these pets.

It's heartening to note that a considerable number of pets find homes within the first week of being listed.

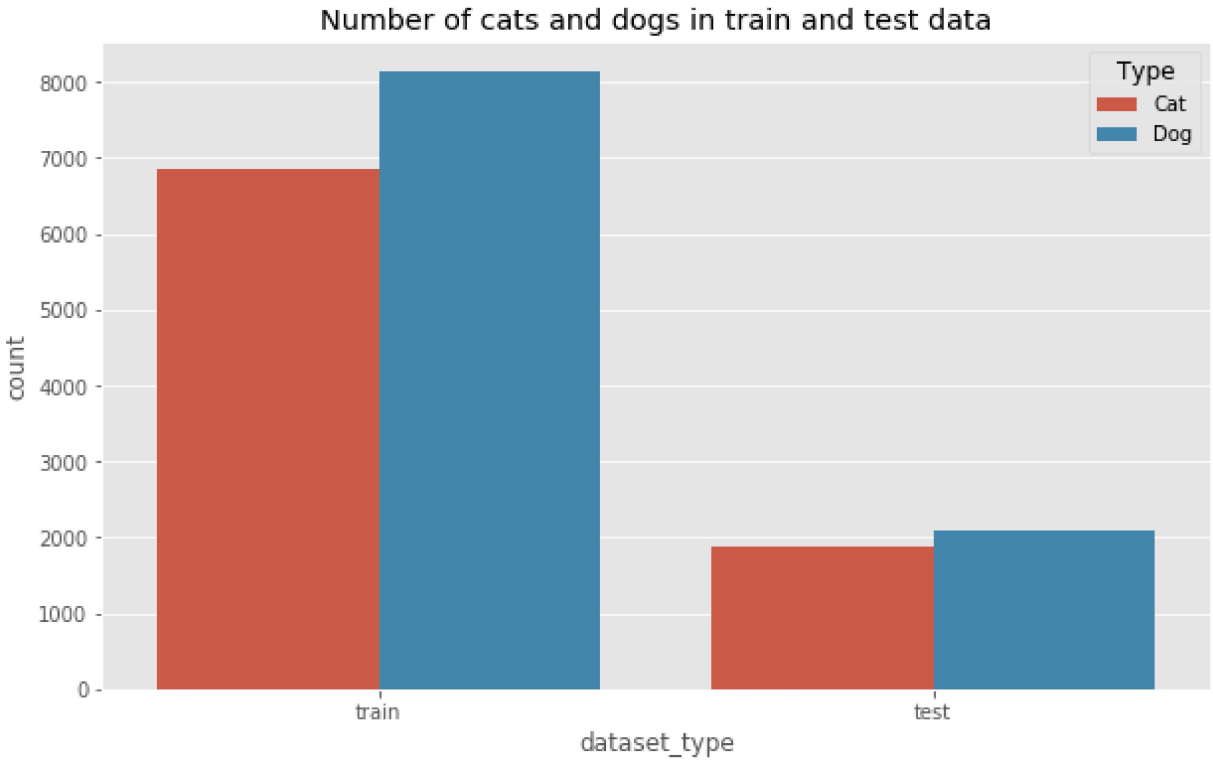
An intriguing observation is the linear relationship among the classes—the higher the assigned number, the more challenging the situation for the pet. This suggests the potential for not only multiclass classification but also regression models.

Type

1 - Dog, 2 - Cat

In [12]:
```
all_data['Type'] = all_data['Type'].apply(lambda x: 'Dog' if x == 1 else 'Cat')
plt.figure(figsize=(10, 6));
```

```
sns.countplot(x='dataset_type', data=all_data, hue='Type');
plt.title('Number of cats and dogs in train and test data');
```



Comparison of rates

As observed earlier, the base rate of pets with Adoption Speed 0 stands at 0.027, calculated as 410 out of 14,993 pets.

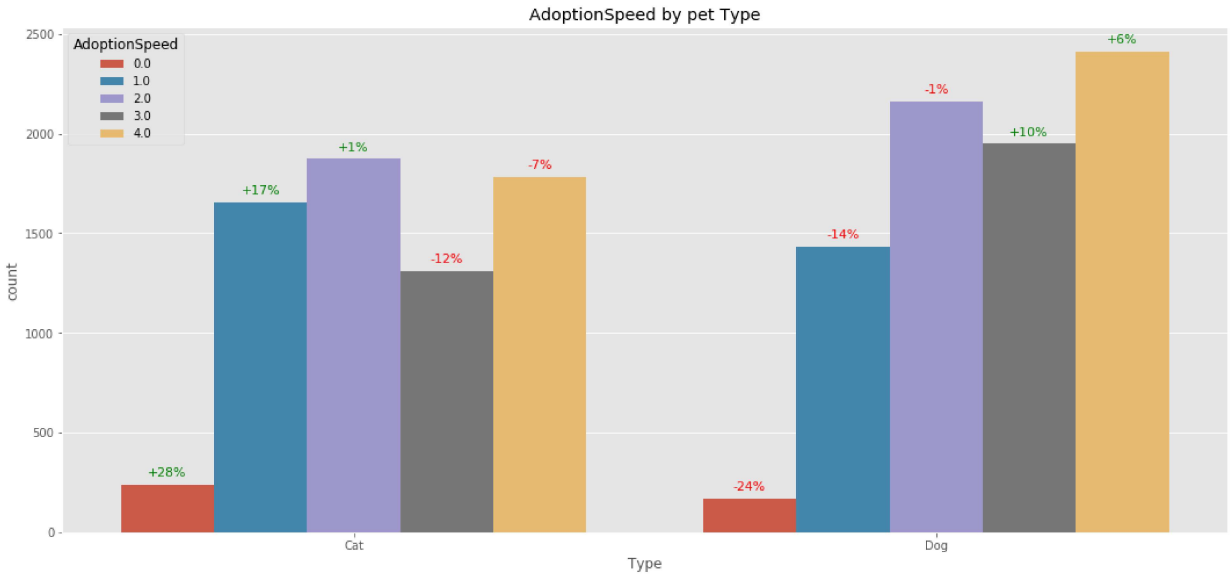
Now, examining the graph specifically for the train dataset, there are 6,861 cats, with 240 of them having Adoption Speed 0. Hence, the rate among cats is 0.035, found by dividing 240 by 6,861.

By comparing the two rates: 0.035 (cat rate) to 0.027 (base rate), we determine a ratio of approximately 1.28. This suggests that cats have a 28% higher likelihood of being in Adoption Speed class 0 compared to the general base rate of adoption, indicating a relatively increased chance for this category among cats as opposed to the overall population of pets.

In [13]:
```
main_count = train['AdoptionSpeed'].value_counts(normalize=True).sort_index()
def prepare_plot_dict(df, col, main_count):
    """
    Preparing dictionary with data for plotting.

    I want to show how much higher/lower are the rates of Adoption speed for the curre
    At first I calculate base rates, then for each category in the column I calculate

    """
    main_count = dict(main_count)
    plot_dict = {}
    for i in df[col].unique():
        val_count = dict(df.loc[df[col] == i, 'AdoptionSpeed'].value_counts().sort_inc
```

```
        for k, v in main_count.items():
            if k in val_count:
                plot_dict[val_count[k]] = ((val_count[k] / sum(val_count.values())) /
            else:
                plot_dict[0] = 0

    return plot_dict

def make_count_plot(df, x, hue='AdoptionSpeed', title='', main_count=main_count):
    """
    Plotting countplot with correct annotations.
    """
    g = sns.countplot(x=x, data=df, hue=hue);
    plt.title(f'AdoptionSpeed {title}');
    ax = g.axes

    plot_dict = prepare_plot_dict(df, x, main_count)

    for p in ax.patches:
        h = p.get_height() if str(p.get_height()) != 'nan' else 0
        text = f"{plot_dict[h]:.0f}%" if plot_dict[h] < 0 else f"+{plot_dict[h]:.0f}%"
        ax.annotate(text, (p.get_x() + p.get_width() / 2., h),
             ha='center', va='center', fontsize=11, color='green' if plot_dict[h] > 0
             textcoords='offset points')
```

In [14]:
```
plt.figure(figsize=(18, 8));
make_count_plot(df=all_data.loc[all_data['dataset_type'] == 'train'], x='Type', title=
```



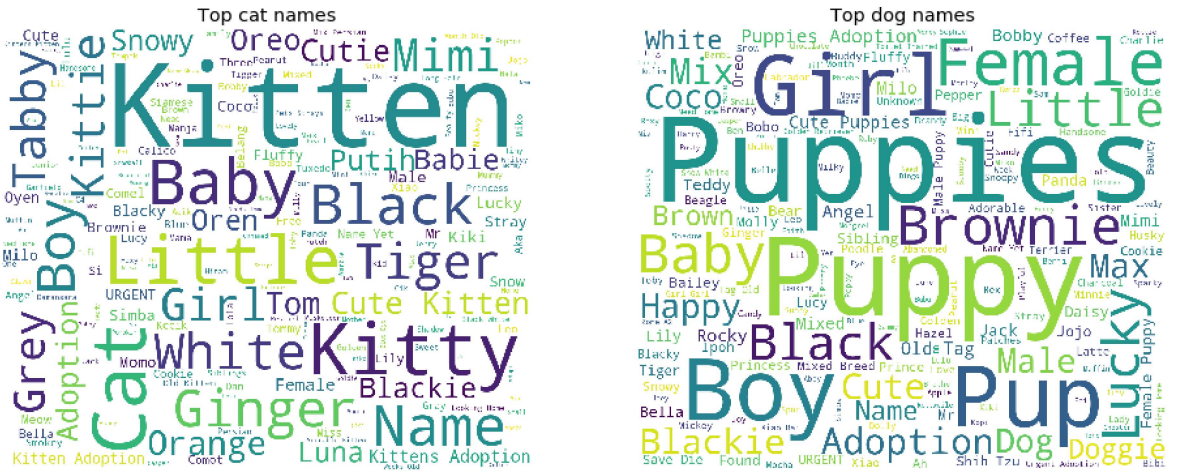
We observe that cats are more likely to be adopted early compared to dogs, and the overall percentage of unadopted cats is lower. Does this suggest a preference for cats? Or could it be that this dataset is small and potentially biased?

Conversely, it appears that more dogs are adopted after several months.

Name

At first let's look at most common names.

In [15]:
```
fig, ax = plt.subplots(figsize = (16, 12))
plt.subplot(1, 2, 1)
text_cat = ' '.join(all_data.loc[all_data['Type'] == 'Cat', 'Name'].fillna('').values)
wordcloud = WordCloud(max_font_size=None, background_color='white',
                      width=1200, height=1000).generate(text_cat)
plt.imshow(wordcloud)
plt.title('Top cat names')
plt.axis("off")

plt.subplot(1, 2, 2)
text_dog = ' '.join(all_data.loc[all_data['Type'] == 'Dog', 'Name'].fillna('').values)
wordcloud = WordCloud(max_font_size=None, background_color='white',
                      width=1200, height=1000).generate(text_dog)
plt.imshow(wordcloud)
plt.title('Top dog names')
plt.axis("off")

plt.show()
```



We frequently encounter typical pet names such as 'Mimi' or 'Angel.' It's common for people to describe the pet for adoption simply as 'Kitten' or 'Puppies.' The pet's color is often mentioned, and sometimes the gender as well. Occasionally, pet names might be unusual, or other information might be provided in place of a name.

Another notable observation is that some pets are without names. Let's explore whether this holds significance.

In [16]:
```
print('Most popular pet names and AdoptionSpeed')
for n in train['Name'].value_counts().index[:5]:
    print(n)
    print(train.loc[train['Name'] == n, 'AdoptionSpeed'].value_counts().sort_index())
    print('')
```

```
Most popular pet names and AdoptionSpeed
Baby
0     2
1    11
2    15
3    11
4    27
Name: AdoptionSpeed, dtype: int64

Lucky
0     5
1    14
2    16
3    12
4    17
Name: AdoptionSpeed, dtype: int64

Brownie
0     1
1    11
2    14
3    12
4    16
Name: AdoptionSpeed, dtype: int64

No Name
0     3
1    14
2    11
3     6
4    20
Name: AdoptionSpeed, dtype: int64

Mimi
0     3
1    12
2    13
3     7
4    17
Name: AdoptionSpeed, dtype: int64
```

No name

In [17]:

```
train['Name'] = train['Name'].fillna('Unnamed')
test['Name'] = test['Name'].fillna('Unnamed')
all_data['Name'] = all_data['Name'].fillna('Unnamed')

train['No_name'] = 0
train.loc[train['Name'] == 'Unnamed', 'No_name'] = 1
test['No_name'] = 0
test.loc[test['Name'] == 'Unnamed', 'No_name'] = 1
all_data['No_name'] = 0
all_data.loc[all_data['Name'] == 'Unnamed', 'No_name'] = 1

print(f"Rate of unnamed pets in train data: {train['No_name'].sum() * 100 / train['No_
print(f"Rate of unnamed pets in test data: {test['No_name'].sum() * 100 / test['No_nam
```

```
Rate of unnamed pets in train data: 8.4173%.
Rate of unnamed pets in test data: 10.3474%.
```

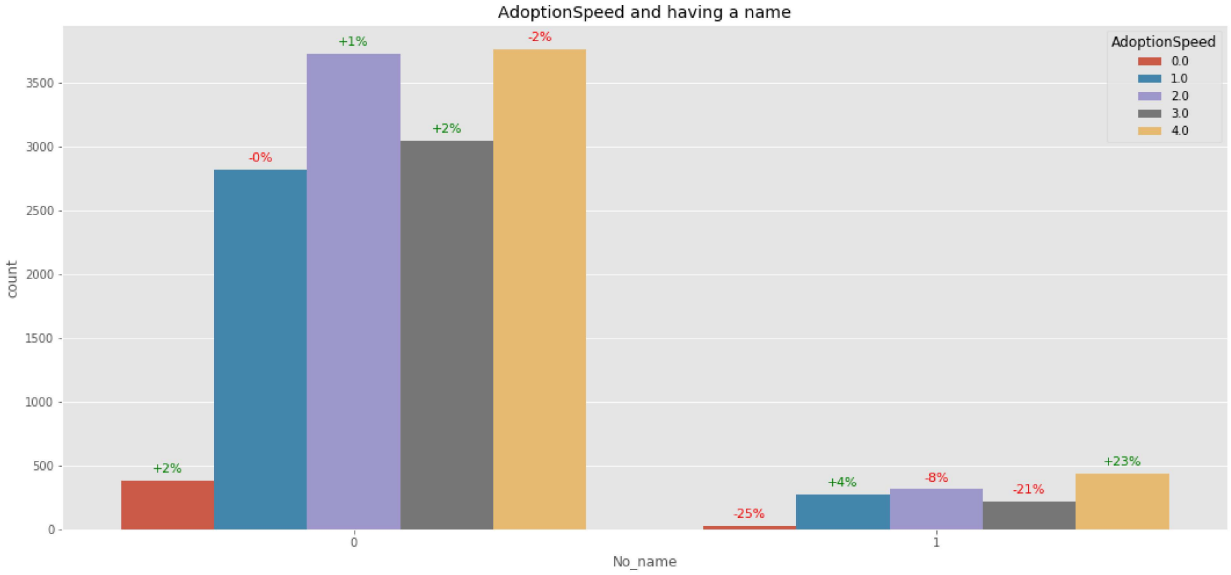
In [18]:
```
pd.crosstab(train['No_name'], train['AdoptionSpeed'], normalize='index')
```

Out[18]:

AdoptionSpeed	0	1	2	3	4
No_name					
0	0.027966	0.205302	0.271211	0.221470	0.274051
1	0.020602	0.214739	0.248019	0.172742	0.343899

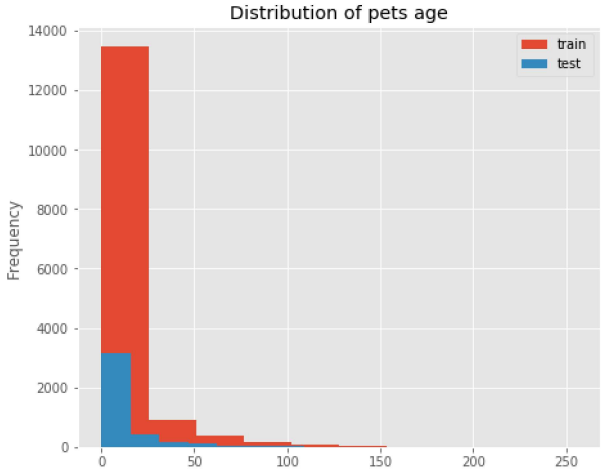
Although less than 10% of pets in the dataset don't have names, they show a higher likelihood of not being adopted.

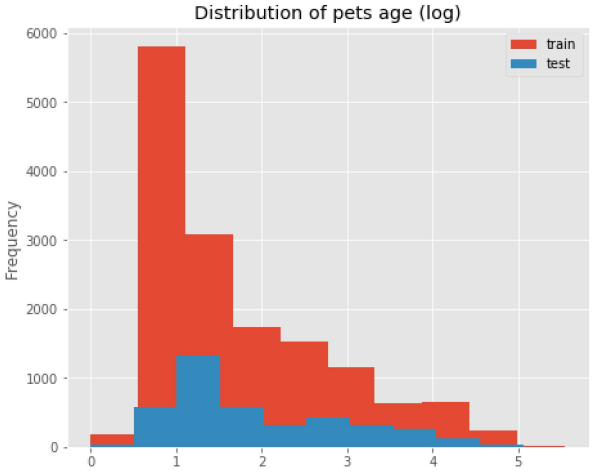
In [19]:
```
plt.figure(figsize=(18, 8));
make_count_plot(df=all_data.loc[all_data['dataset_type'] == 'train'], x='No_name', tit
```



Age

In [22]:
```
fig, ax = plt.subplots(figsize = (16, 6))
plt.subplot(1, 2, 1)
plt.title('Distribution of pets age');
train['Age'].plot('hist', label='train');
test['Age'].plot('hist', label='test');
plt.legend();

plt.subplot(1, 2, 2)
plt.title('Distribution of pets age (log)');
np.log1p(train['Age']).plot('hist', label='train');
np.log1p(test['Age']).plot('hist', label='test');
plt.legend();
```



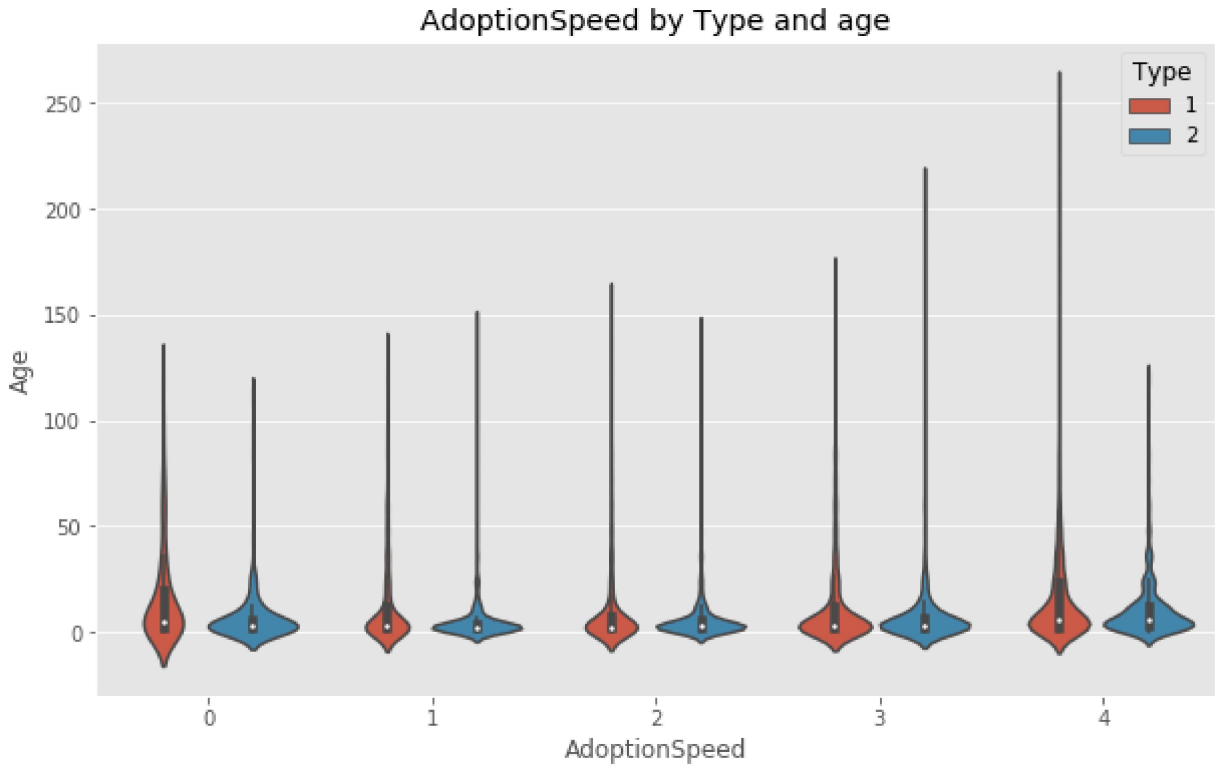


In [23]:
```
train['Age'].value_counts().head(10)
```

Out[23]:
```
2     3503
1     2304
3     1966
4     1109
12     967
24     651
5      595
6      558
36     417
8      309
Name: Age, dtype: int64
```

It's evident that the majority of pets are young, possibly indicating their age post-birth. Additionally, there are numerous pets with ages that are multiples of 12. This suggests that owners might not have provided precise ages for these pets.

In [24]:
```
plt.figure(figsize=(10, 6));
sns.violinplot(x="AdoptionSpeed", y="Age", hue="Type", data=train);
plt.title('AdoptionSpeed by Type and age');
```



Breeds

The pets have a primary breed, and if applicable, a secondary breed.

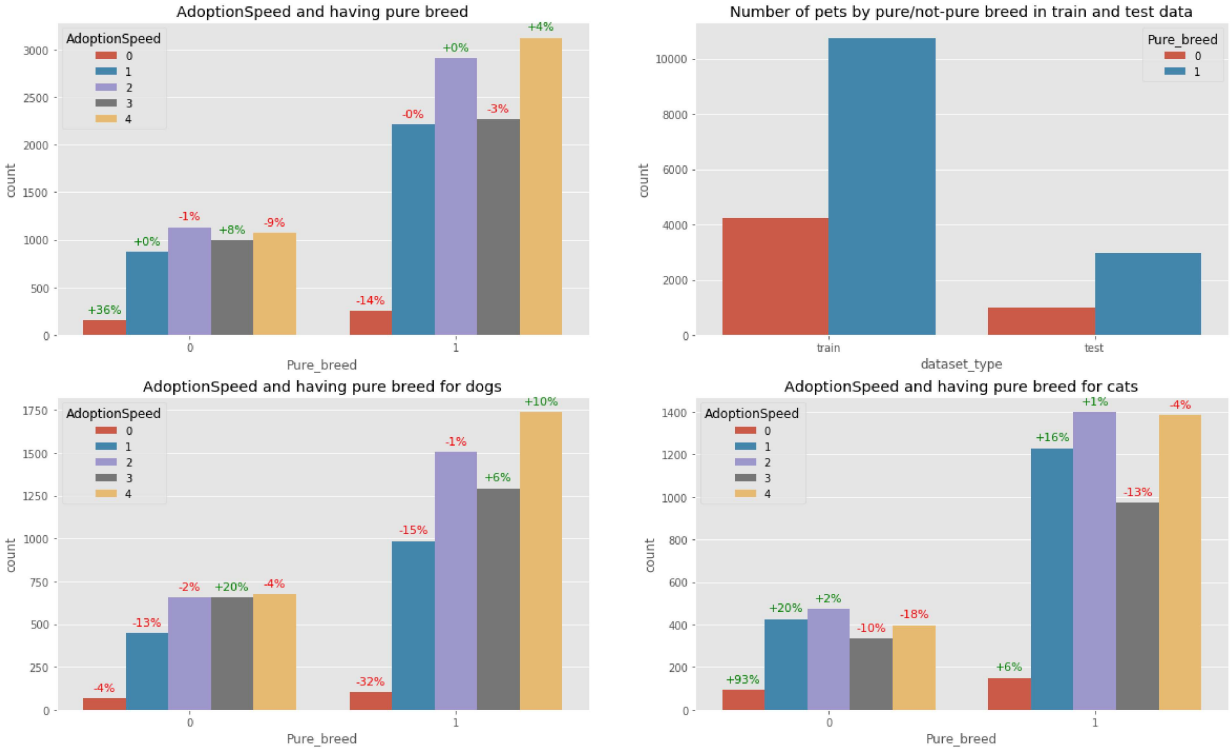
Initially, let's examine whether having a secondary breed influences the adoption speed.

In [26]:
```
train['Pure_breed'] = 0
train.loc[train['Breed2'] == 0, 'Pure_breed'] = 1
test['Pure_breed'] = 0
test.loc[test['Breed2'] == 0, 'Pure_breed'] = 1
all_data['Pure_breed'] = 0
all_data.loc[all_data['Breed2'] == 0, 'Pure_breed'] = 1

print(f"Rate of pure breed pets in train data: {train['Pure_breed'].sum() * 100 / trai
print(f"Rate of pure breed pets in test data: {test['Pure_breed'].sum() * 100 / test['
```

```
Rate of pure breed pets in train data: 71.7802%.
Rate of pure breed pets in test data: 74.6979%.
```

In [27]:
```
def plot_four_graphs(col='', main_title='', dataset_title=''):
    """
    Plotting four graphs:
    - adoption speed by variable;
    - counts of categories in the variable in train and test;
    - adoption speed by variable for dogs;
    - adoption speed by variable for cats;
    """
    plt.figure(figsize=(20, 12));
    plt.subplot(2, 2, 1)
    make_count_plot(df=train, x=col, title=f'and {main_title}')

    plt.subplot(2, 2, 2)
    sns.countplot(x='dataset_type', data=all_data, hue=col);
```

```
    plt.title(dataset_title);

    plt.subplot(2, 2, 3)
    make_count_plot(df=train.loc[train['Type'] == 1], x=col, title=f'and {main_title}

    plt.subplot(2, 2, 4)
    make_count_plot(df=train.loc[train['Type'] == 2], x=col, title=f'and {main_title}

plot_four_graphs(col='Pure_breed', main_title='having pure breed', dataset_title='Numb
```



It appears that mixed-breed pets, especially cats, tend to be adopted more frequently and at a faster rate.

Let's delve deeper into the specific breeds themselves.

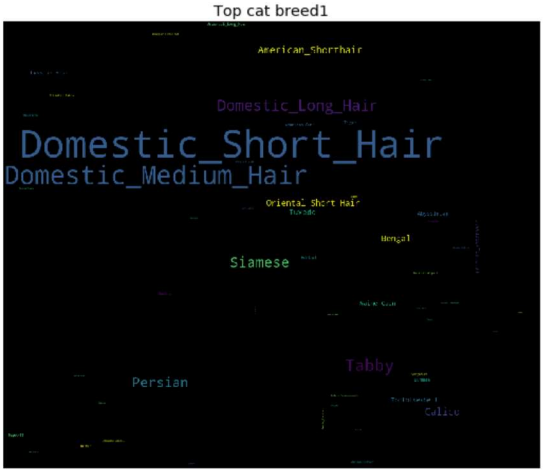
In [28]:
```
breeds_dict = {k: v for k, v in zip(breeds['BreedID'], breeds['BreedName'])}
```

In [29]:
```
train['Breed1_name'] = train['Breed1'].apply(lambda x: '_'.join(breeds_dict[x].split()
train['Breed2_name'] = train['Breed2'].apply(lambda x: '_'.join(breeds_dict[x]) if x i

test['Breed1_name'] = test['Breed1'].apply(lambda x: '_'.join(breeds_dict[x].split())
test['Breed2_name'] = test['Breed2'].apply(lambda x: '_'.join(breeds_dict[x].split())

all_data['Breed1_name'] = all_data['Breed1'].apply(lambda x: '_'.join(breeds_dict[x].s
all_data['Breed2_name'] = all_data['Breed2'].apply(lambda x: '_'.join(breeds_dict[x].s
```

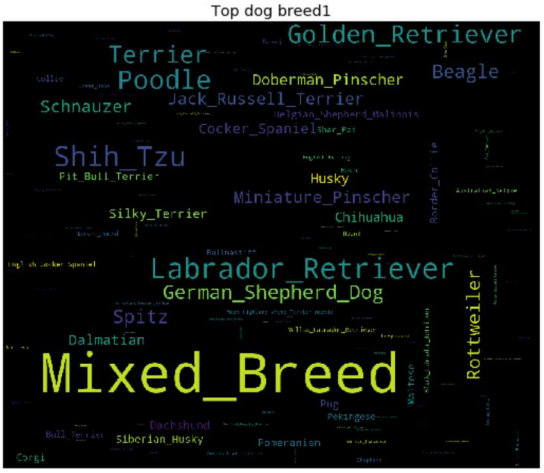
In [30]:
```
fig, ax = plt.subplots(figsize = (20, 18))
plt.subplot(2, 2, 1)
text_cat1 = ' '.join(all_data.loc[all_data['Type'] == 'Cat', 'Breed1_name'].fillna('')
wordcloud = WordCloud(max_font_size=None, background_color='black', collocations=False
                      width=1200, height=1000).generate(text_cat1)
plt.imshow(wordcloud)
plt.title('Top cat breed1')
plt.axis("off")
```

```
plt.subplot(2, 2, 2)
text_dog1 = ' '.join(all_data.loc[all_data['Type'] == 'Dog', 'Breed1_name'].fillna('')
wordcloud = WordCloud(max_font_size=None, background_color='black', collocations=False
                      width=1200, height=1000).generate(text_dog1)
plt.imshow(wordcloud)
plt.title('Top dog breed1')
plt.axis("off")

plt.subplot(2, 2, 3)
text_cat2 = ' '.join(all_data.loc[all_data['Type'] == 'Cat', 'Breed2_name'].fillna('')
wordcloud = WordCloud(max_font_size=None, background_color='black', collocations=False
                      width=1200, height=1000).generate(text_cat2)
plt.imshow(wordcloud)
plt.title('Top cat breed1')
plt.axis("off")

plt.subplot(2, 2, 4)
text_dog2 = ' '.join(all_data.loc[all_data['Type'] == 'Dog', 'Breed2_name'].fillna('')
wordcloud = WordCloud(max_font_size=None, background_color='black', collocations=False
                      width=1200, height=1000).generate(text_dog2)
plt.imshow(wordcloud)
plt.title('Top dog breed2')
plt.axis("off")
plt.show()
```

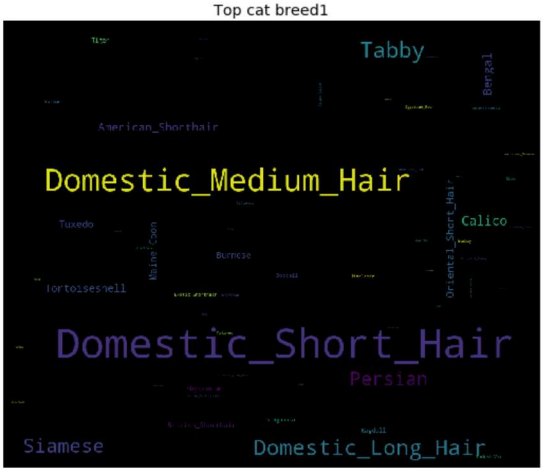
Top cat breed1



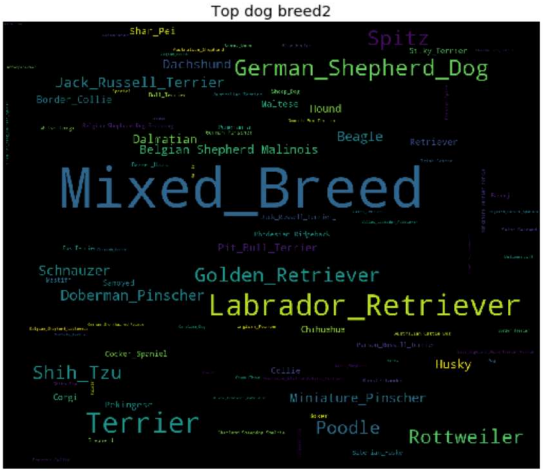
Top dog breed1



Top cat breed1



Top dog breed2



It appears that not all values within these features truly represent specific breeds. At times, individuals simply indicate that the dogs have a mixed breed, while cats are often described as domestic with a specific hair length.

Now, let's explore the combinations of breed names to gain further insights.

In [31]:
```
(all_data['Breed1_name'] + '__' + all_data['Breed2_name']).value_counts().head(15)
```

Out[31]:
```
Mixed_Breed__-                                5724
Domestic_Short_Hair__-                        3945
Domestic_Medium_Hair__-                       1255
Mixed_Breed__Mixed_Breed                      1213
Domestic_Short_Hair__Domestic_Short_Hair       330
Tabby__-                                       294
Domestic_Long_Hair__-                          220
Shih_Tzu__-                                    203
Poodle__-                                      162
Siamese__-                                     147
Labrador_Retriever__Mixed_Breed                146
Domestic_Medium_Hair__Domestic_Medium_Hair     119
Golden_Retriever__-                            119
Domestic_Medium_Hair__Domestic_Short_Hair      114
Persian__-                                     108
dtype: int64
```

It appears that the majority of dogs aren't pure breeds but rather mixed breeds. My initial assumption was incorrect.

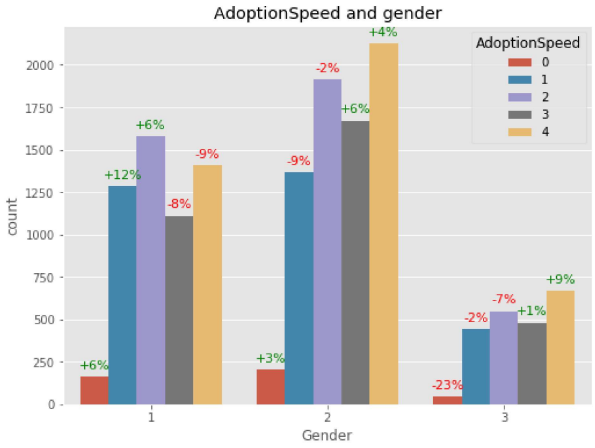
Interestingly, there's variability in the data representation. Sometimes 'mixed breed' is written in the first field, sometimes in both, and occasionally, the main breed is listed in the first field while being marked as 'mixed breed' in the second field.

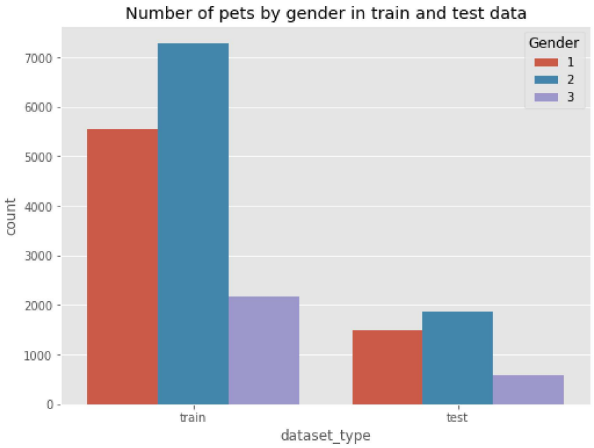
I believe we can generate new features based on this information. Subsequently, we can further verify the hair length of pets.

Gender

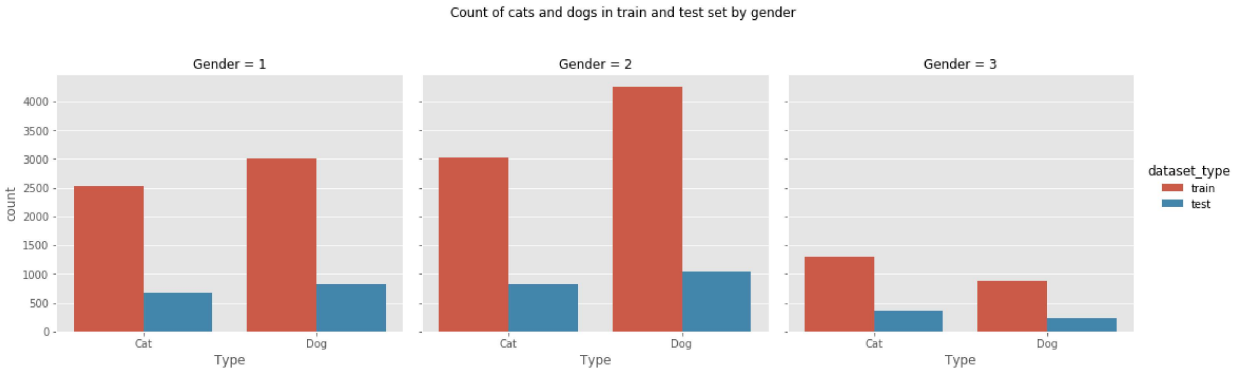
1 = Male, 2 = Female, 3 = Mixed

In [32]:
```
plt.figure(figsize=(18, 6));
plt.subplot(1, 2, 1)
make_count_plot(df=train, x='Gender', title='and gender')

plt.subplot(1, 2, 2)
sns.countplot(x='dataset_type', data=all_data, hue='Gender');
plt.title('Number of pets by gender in train and test data');
```





In [33]:
```
sns.factorplot('Type', col='Gender', data=all_data, kind='count', hue='dataset_type');
plt.subplots_adjust(top=0.8)
plt.suptitle('Count of cats and dogs in train and test set by gender');
```



It appears that male pets are adopted more quickly than females. The absence of gender information significantly decreases the chances of adoption.

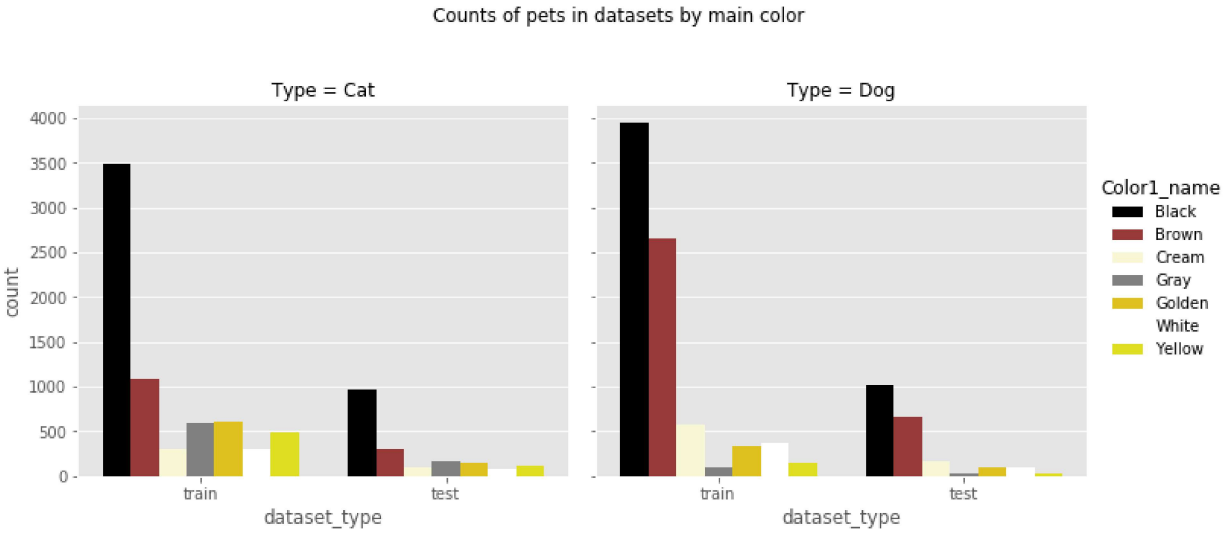
Colors

In [34]:
```
colors_dict = {k: v for k, v in zip(colors['ColorID'], colors['ColorName'])}
train['Color1_name'] = train['Color1'].apply(lambda x: colors_dict[x] if x in colors_d
train['Color2_name'] = train['Color2'].apply(lambda x: colors_dict[x] if x in colors_d
train['Color3_name'] = train['Color3'].apply(lambda x: colors_dict[x] if x in colors_d

test['Color1_name'] = test['Color1'].apply(lambda x: colors_dict[x] if x in colors_dic
test['Color2_name'] = test['Color2'].apply(lambda x: colors_dict[x] if x in colors_dic
test['Color3_name'] = test['Color3'].apply(lambda x: colors_dict[x] if x in colors_dic

all_data['Color1_name'] = all_data['Color1'].apply(lambda x: colors_dict[x] if x in co
all_data['Color2_name'] = all_data['Color2'].apply(lambda x: colors_dict[x] if x in co
all_data['Color3_name'] = all_data['Color3'].apply(lambda x: colors_dict[x] if x in co
```

In [35]:
```
def make_factor_plot(df, x, col, title, main_count=main_count, hue=None, ann=True, col
    """
    Plotting countplot.
    Making annotations is a bit more complicated, because we need to iterate over axes
    """
    if hue:
        g = sns.factorplot(col, col=x, data=df, kind='count', col_wrap=col_wrap, hue=h
    else:
```

```
        g = sns.factorplot(col, col=x, data=df, kind='count', col_wrap=col_wrap);
    plt.subplots_adjust(top=0.9);
    plt.suptitle(title);
    ax = g.axes
    plot_dict = prepare_plot_dict(df, x, main_count)
    if ann:
        for a in ax:
            for p in a.patches:
                text = f"{plot_dict[p.get_height()]:.0f}%" if plot_dict[p.get_height()
                a.annotate(text, (p.get_x() + p.get_width() / 2., p.get_height()),
                     ha='center', va='center', fontsize=11, color='green' if plot_dict
                     textcoords='offset points')
```

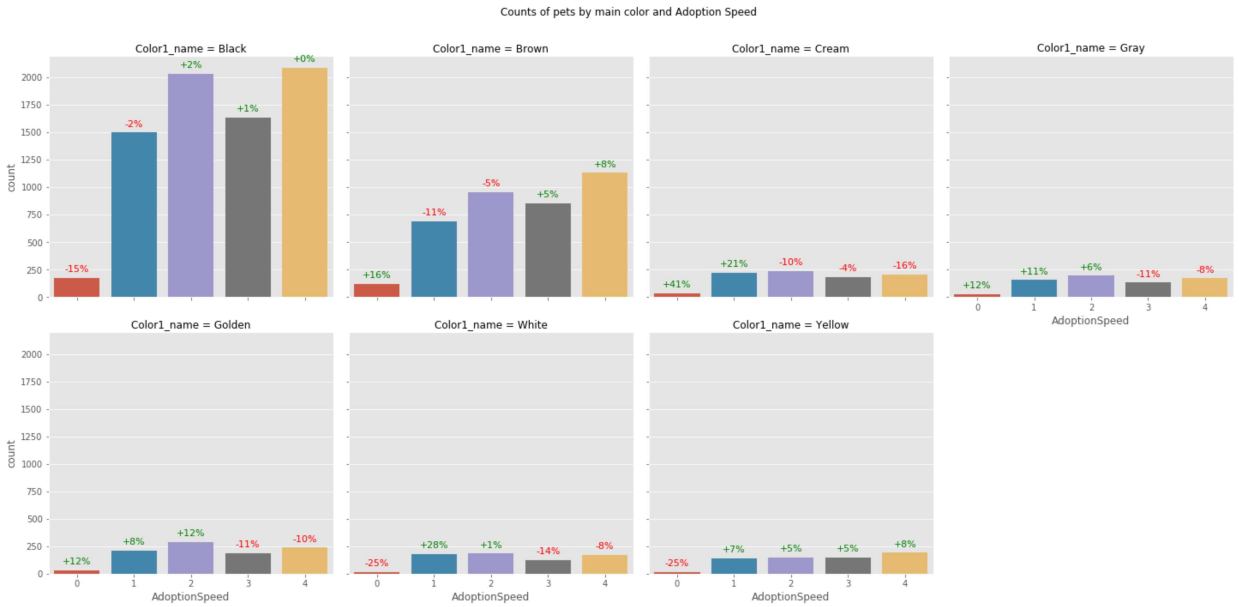
In [36]:
```
sns.factorplot('dataset_type', col='Type', data=all_data, kind='count', hue='Color1_na
plt.subplots_adjust(top=0.8)
plt.suptitle('Counts of pets in datasets by main color');
```



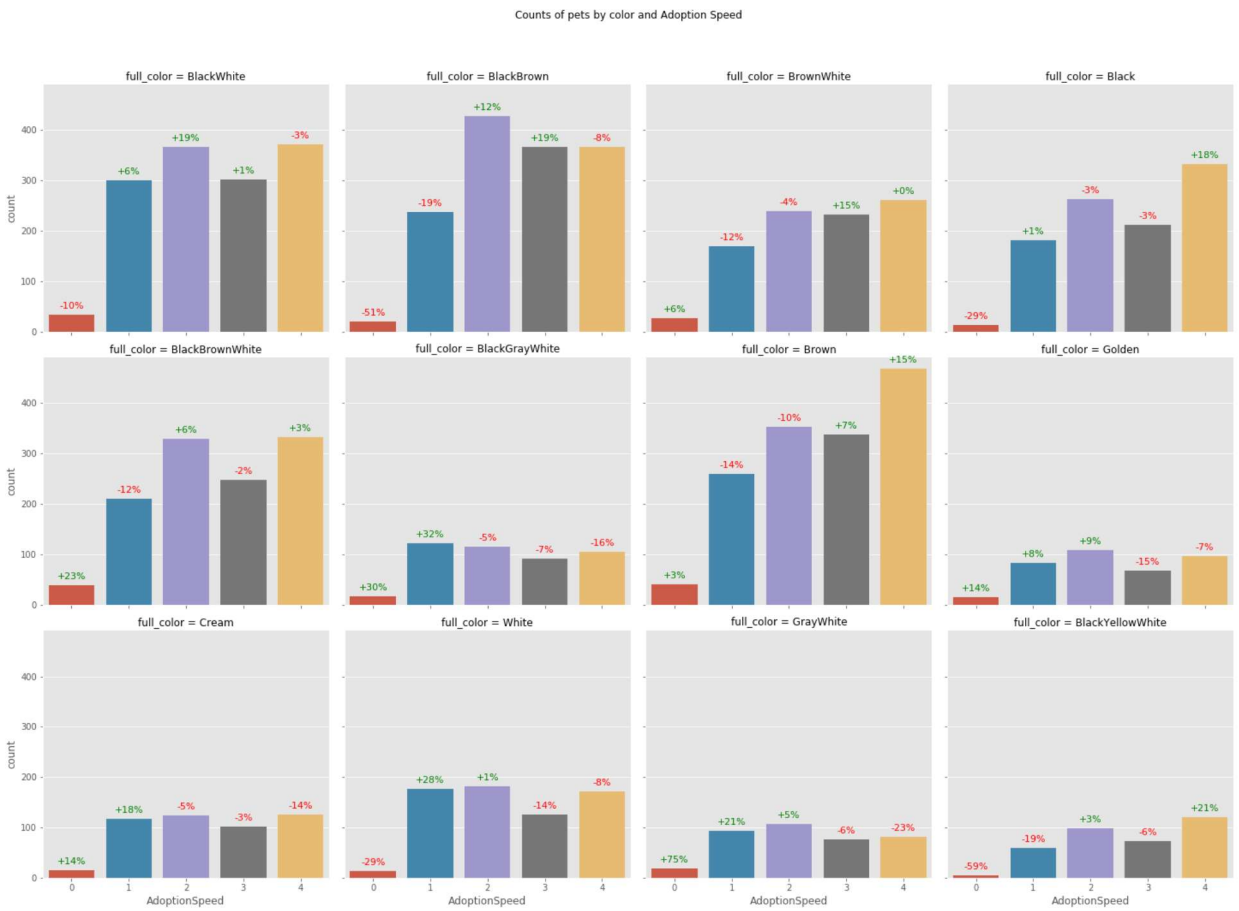
We observe that the most common colors are black and brown. It's interesting to note the scarcity of gray or yellow dogs.

Next, let's explore whether colors have an influence on adoption speed.

In [37]:
```
make_factor_plot(df=train, x='Color1_name', col='AdoptionSpeed', title='Counts of pets
```



In [38]:
```
train['full_color'] = (train['Color1_name'] + '__' + train['Color2_name'] + '__' + tra
test['full_color'] = (test['Color1_name'] + '__' + test['Color2_name'] + '__' + test['
all_data['full_color'] = (all_data['Color1_name'] + '__' + all_data['Color2_name'] + '

make_factor_plot(df=train.loc[train['full_color'].isin(list(train['full_color'].value_
```



While we notice variations based on color, it's important to note that the number of pets in most colors isn't notably high. Therefore, these differences might be due to randomness.

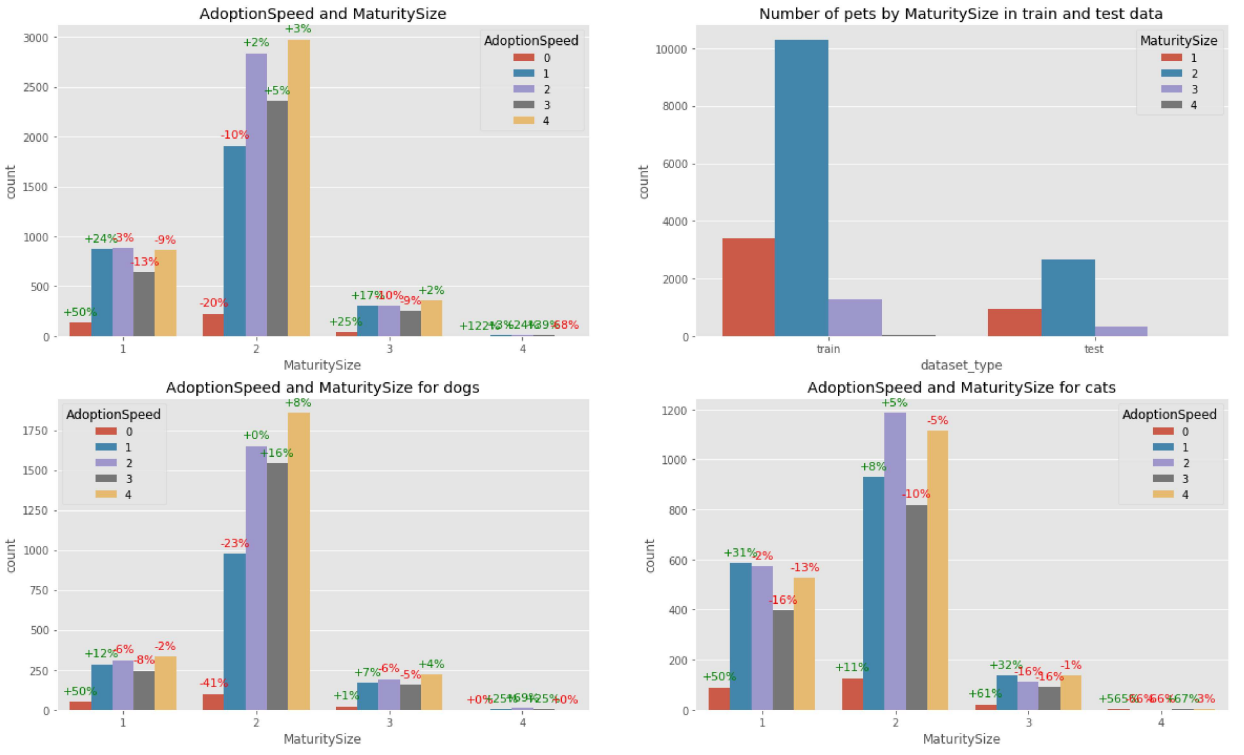
In [39]:
```
gender_dict = {1: 'Male', 2: 'Female', 3: 'Mixed'}
for i in all_data['Type'].unique():
    for j in all_data['Gender'].unique():
        df = all_data.loc[(all_data['Type'] == i) & (all_data['Gender'] == j)]
        top_colors = list(df['full_color'].value_counts().index)[:5]
        j = gender_dict[j]
        print(f"Most popular colors of {j} {i}s: {' '.join(top_colors)}")
```

```
Most popular colors of Male Cats: BlackWhite Black GrayWhite White YellowWhite
Most popular colors of Female Cats: BlackWhite BlackBrownWhite BlackYellowWhite GrayW
hite White
Most popular colors of Mixed Cats: BlackBrownWhite BlackGrayWhite BlackYellowWhite Bl
ackWhite BlackGoldenWhite
Most popular colors of Male Dogs: Brown BlackBrown BrownWhite Black BlackWhite
Most popular colors of Female Dogs: Brown BlackBrown BrownWhite Black BlackWhite
Most popular colors of Mixed Dogs: BlackBrownWhite BlackBrown BlackBrownCream Brown B
lackWhite
```

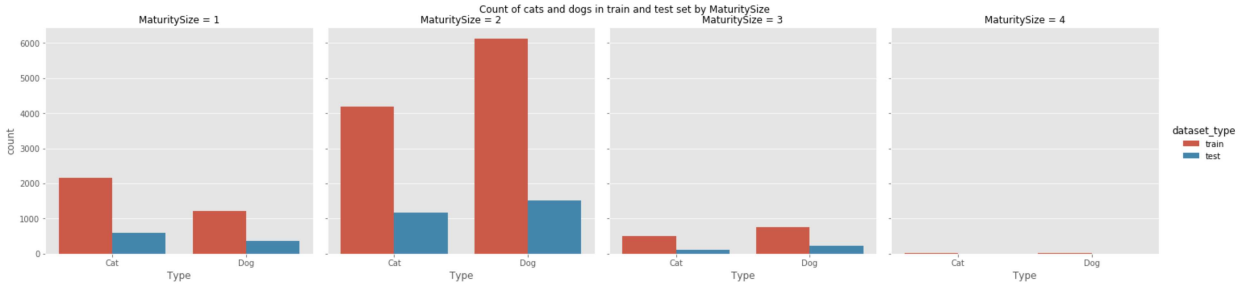
Matirity Size

Size at maturity (1 = Small, 2 = Medium, 3 = Large, 4 = Extra Large, 0 = Not Specified)

In [40]:
```
plot_four_graphs(col='MaturitySize', main_title='MaturitySize', dataset_title='Number
```



In [41]:
```
make_factor_plot(df=all_data, x='MaturitySize', col='Type', title='Count of cats and d
```



In [42]:
```
images = [i.split('-')[0] for i in os.listdir('../input/train_images/')]
size_dict = {1: 'Small', 2: 'Medium', 3: 'Large', 4: 'Extra Large'}
for t in all_data['Type'].unique():
    for m in all_data['MaturitySize'].unique():
        df = all_data.loc[(all_data['Type'] == t) & (all_data['MaturitySize'] == m)]
        top_breeds = list(df['Breed1_name'].value_counts().index)[:5]
        m = size_dict[m]
        print(f"Most common Breeds of {m} {t}s:")

        fig = plt.figure(figsize=(25, 4))

        for i, breed in enumerate(top_breeds):
            # excluding pets without pictures
            b_df = df.loc[(df['Breed1_name'] == breed) & (df['PetID'].isin(images)), '
            if len(b_df) > 1:
                pet_id = b_df.values[1]
            else:
                pet_id = b_df.values[0]
            ax = fig.add_subplot(1, 5, i+1, xticks=[], yticks=[])

            im = Image.open("../input/train_images/" + pet_id + '-1.jpg')
            plt.imshow(im)
            ax.set_title(f'Breed: {breed}')
        plt.show();
```

Most common Breeds of Small Cats:

Most common Breeds of Medium Cats:

Most common Breeds of Large Cats:

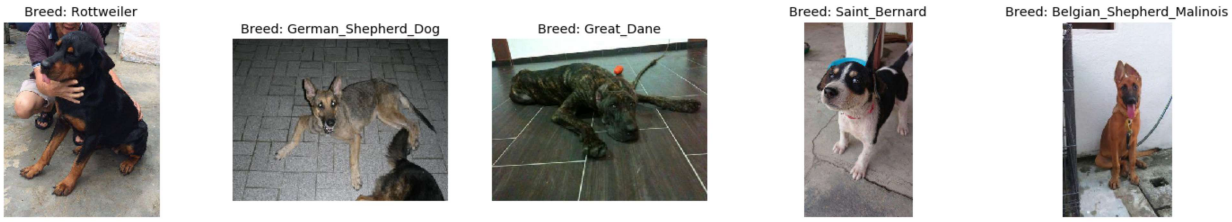
Most common Breeds of Extra Large Cats:

Most common Breeds of Small Dogs:

Most common Breeds of Medium Dogs:

Most common Breeds of Large Dogs:

Most common Breeds of Extra Large Dogs:



We can observe that maturity size doesn't appear to be significantly influential. Medium-sized pets are the most common, and they slightly have higher chances of not being adopted.

There is a scarcity of Extra Large pets. Hopefully, this indicates their owners cherish them, and there might be no urgent need for them to be adopted.

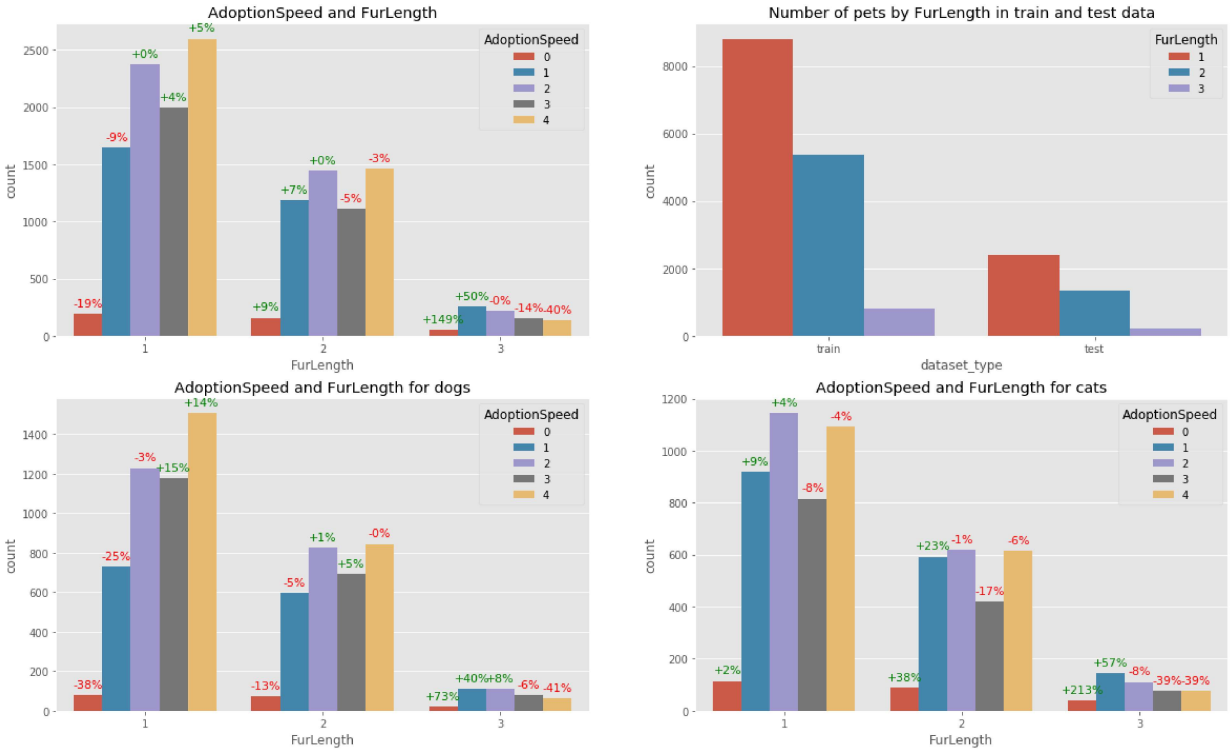
To showcase a variety of pets, I presented examples of pictures featuring the most common breeds for each maturity size of both cats and dogs.

I suspect that not all the data is entirely accurate. Sometimes short-haired cats are labeled with breeds featuring 'medium hair,' raising doubts about the accuracy of all breed classifications. Furthermore, some photos might have poor quality.

FurLength

(1 = Short, 2 = Medium, 3 = Long, 0 = Not Specified)

In [43]:
```
plot_four_graphs(col='FurLength', main_title='FurLength', dataset_title='Number of pet
```



We observe that the majority of pets have short fur, while long fur is the least common.

Pets with long hair seem to have a higher chance of being adopted. However, this observation could potentially be due to randomness resulting from the lower count of long-haired pets.

As mentioned earlier, some breeds are identified with specific hair lengths in the text. Let's further examine these values for accuracy.

In [45]:
```
c = 0
strange_pets = []
for i, row in all_data[all_data['Breed1_name'].str.contains('air')].iterrows():
    if 'Short' in row['Breed1_name'] and row['FurLength'] == 1:
        pass
    elif 'Medium' in row['Breed1_name'] and row['FurLength'] == 2:
        pass
    elif 'Long' in row['Breed1_name'] and row['FurLength'] == 3:
        pass
    else:
        c += 1
        strange_pets.append((row['PetID'], row['Breed1_name'], row['FurLength']))

print(f"There are {c} pets whose breed and fur length don't match")
```

```
There are 921 pets whose breed and fur length don't match
```

In [46]:
```
strange_pets = [p for p in strange_pets if p[0] in images]
fig = plt.figure(figsize=(25, 12))
fur_dict = {1: 'Short', 2: 'Medium', 3: 'long'}
for i, s in enumerate(random.sample(strange_pets, 12)):
    ax = fig.add_subplot(3, 4, i+1, xticks=[], yticks=[])
```

```
    im = Image.open("../input/train_images/" + s[0] + '-1.jpg')
    plt.imshow(im)
    ax.set_title(f'Breed: {s[1]} \n Fur length: {fur_dict[s[2]]}')
plt.show();
```

At times, the breed is more accurate, while at other times, it's the fur length indicated in the data. I believe it would be beneficial to create a feature that indicates whether the breed and fur length information align or match.

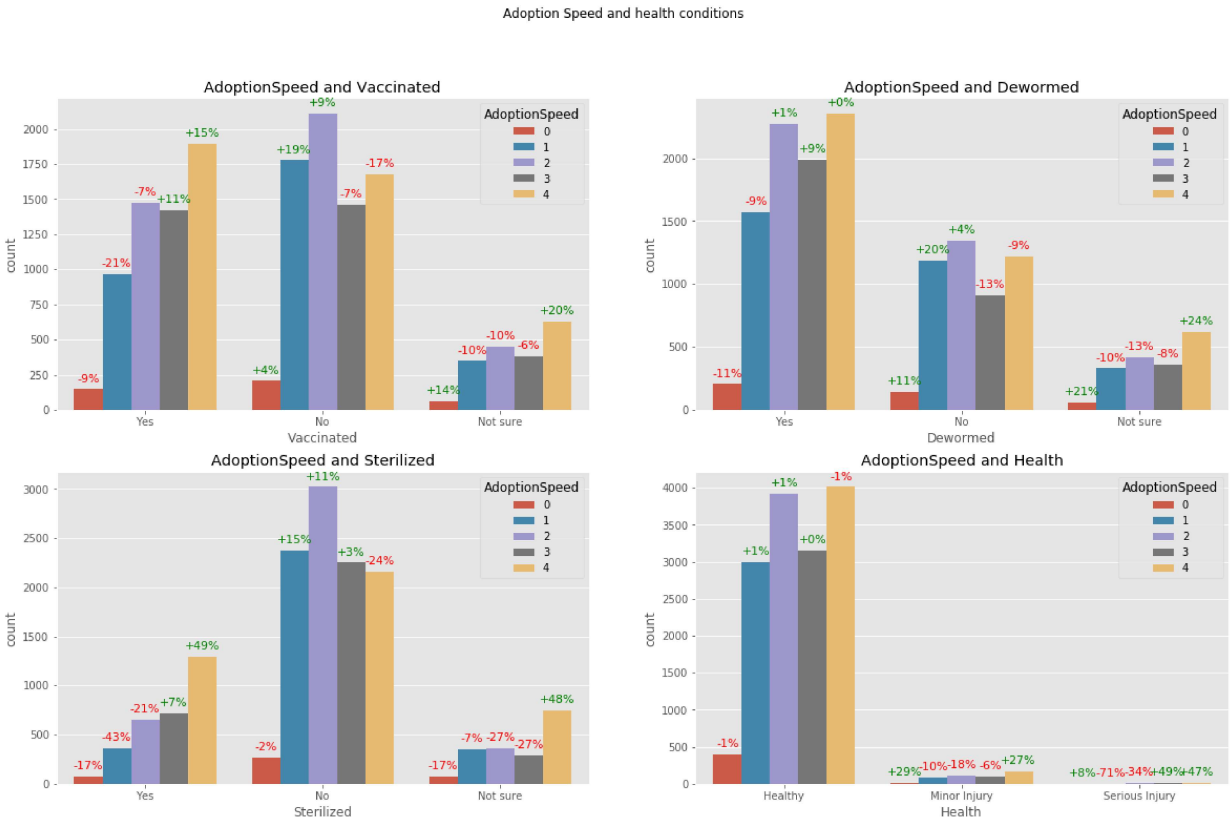
Health

There are four features indicating the health status of the pets:

Vaccinated - Indicates whether the pet has been vaccinated (1 = Yes, 2 = No, 3 = Not Sure) Dewormed - Shows if the pet has been dewormed (1 = Yes, 2 = No, 3 = Not Sure) Sterilized - Reflects whether the pet has been spayed or neutered (1 = Yes, 2 = No, 3 = Not Sure) Health - Indicates the health condition of the pet (1 = Healthy, 2 = Minor Injury, 3 = Serious Injury, 0 = Not Specified) I believe these features are crucial; most people would prefer a healthy pet. While sterilization might not be the primary concern, having a healthy and dewormed pet should be of significant importance. Let's analyze the data to confirm these assumptions.

In [47]:
```
plt.figure(figsize=(20, 12));
plt.subplot(2, 2, 1)
make_count_plot(df=train, x='Vaccinated', title='Vaccinated')
plt.xticks([0, 1, 2], ['Yes', 'No', 'Not sure']);
plt.title('AdoptionSpeed and Vaccinated');

plt.subplot(2, 2, 2)
make_count_plot(df=train, x='Dewormed', title='Dewormed')
plt.xticks([0, 1, 2], ['Yes', 'No', 'Not sure']);
plt.title('AdoptionSpeed and Dewormed');

plt.subplot(2, 2, 3)
make_count_plot(df=train, x='Sterilized', title='Sterilized')
```

```
plt.xticks([0, 1, 2], ['Yes', 'No', 'Not sure']);
plt.title('AdoptionSpeed and Sterilized');

plt.subplot(2, 2, 4)
make_count_plot(df=train, x='Health', title='Health')
plt.xticks([0, 1, 2], ['Healthy', 'Minor Injury', 'Serious Injury']);
plt.title('AdoptionSpeed and Health');

plt.suptitle('Adoption Speed and health conditions');
```



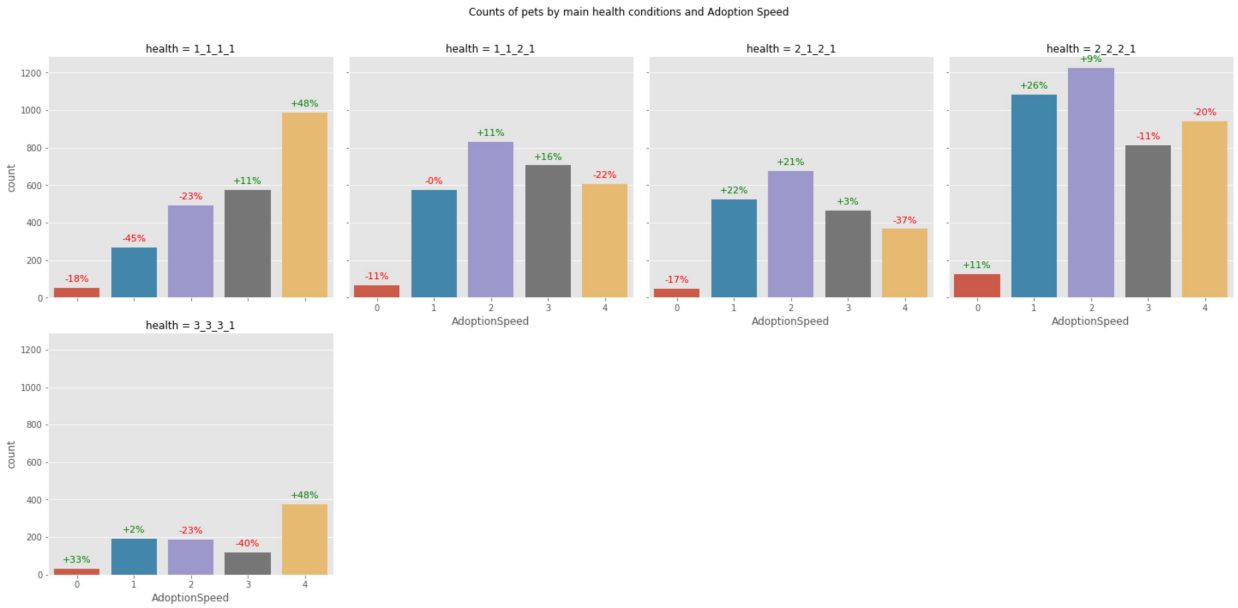
Most pets are in good health, with minor injuries being rare and unfortunately having lower adoption rates. The number of pets with serious injuries is negligible.

Interestingly, there is a preference for non-vaccinated and non-sterilized pets. It's possible that individuals prefer to take pets to vets themselves or want puppies/kittens specifically.

An important observation is that when there's no information about the pet's health condition, the probability of not being adopted increases significantly.

Let's delve into the analysis of the most prevalent health conditions.

In [48]:
```
train['health'] = train['Vaccinated'].astype(str) + '_' + train['Dewormed'].astype(str
test['health'] = test['Vaccinated'].astype(str) + '_' + test['Dewormed'].astype(str) +


make_factor_plot(df=train.loc[train['health'].isin(list(train.health.value_counts().ir
```

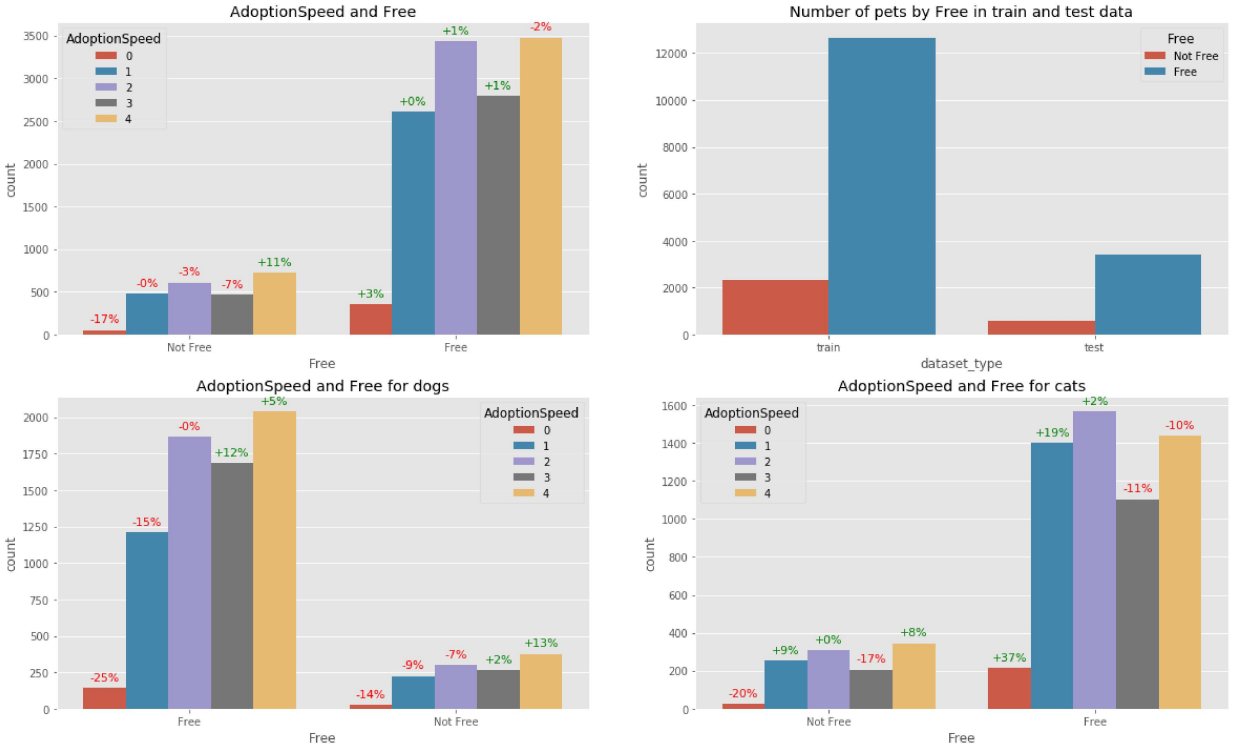


Healthy, dewormed, and non-sterilized pets exhibit faster adoption rates. Surprisingly, completely healthy pets are more likely to remain unadopted. This suggests that many people may prioritize other characteristics over perfect health. Additionally, healthy pets with no information (designated as 'not sure' values) also tend to have lower adoption rates. This could imply that people prefer having comprehensive information, even if it includes negative aspects.

Fee

One interesting feature is the adoption fee. Some pets can be obtained for free, while adopting others requires paying a certain amount.

In [53]:
```
train['Free'] = train['Fee'].apply(lambda x: 'Free' if x == 0 else 'Not Free')
test['Free'] = test['Fee'].apply(lambda x: 'Free' if x == 0 else 'Not Free')
all_data['Free'] = all_data['Fee'].apply(lambda x: 'Free' if x == 0 else 'Not Free')
plot_four_graphs(col='Free', main_title='Free', dataset_title='Number of pets by Free
```



Most pets are available for free, and it appears that requesting a fee slightly decreases the chance of adoption. Notably, free cats are adopted more quickly compared to free dogs.

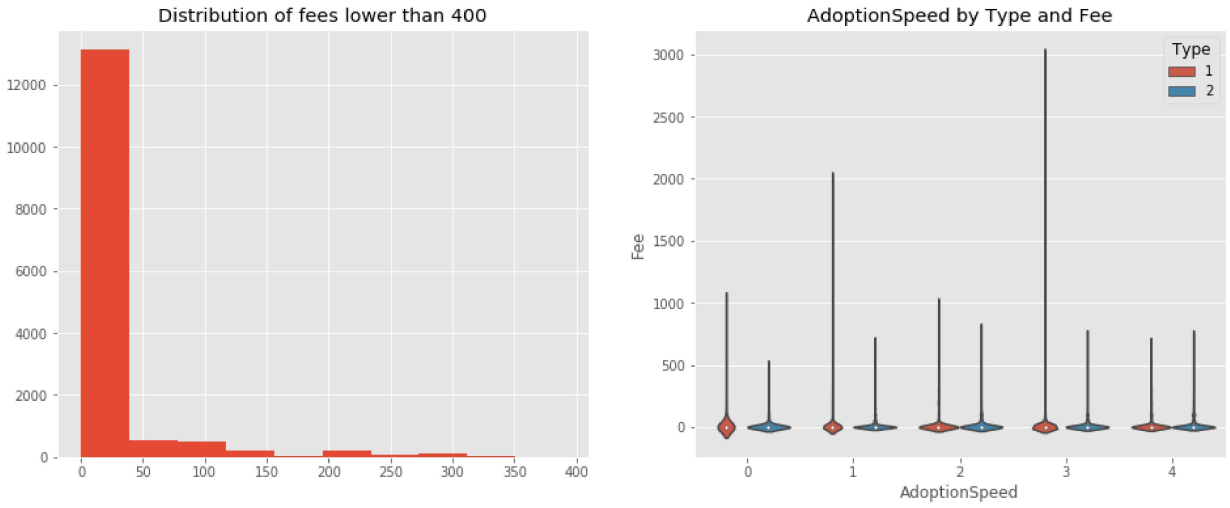
In [54]:
```
all_data.sort_values('Fee', ascending=False)[['Name', 'Description', 'Fee', 'AdoptionS
```

Out[54]:

	Name	Description	Fee	AdoptionSpeed	dataset_type
8722	Khaleesi And Drogo	Both pups are family home trained. They love their walks on lead and off lead. Both male and female are from different lineage and both have MKA Reg Certs. They are 4mths and at perfect age for training. The 2pups know some commands like "sit" "outside" "leave" "crate" and more. They are family friendly and can stay/sleep indoors and out. They've just learnt to swim and belly rubs. Great dogs for families that wants a pet and a watchdog too. All vaccines complete, dewormed, microchipped and ...	3000	3.0	train
10477	Bull Dog	Found this bull dog near my neighbourhood for a month now. I have 3 dogs myself at home. Cannot take care of it. Hopefully if there is someone who is loving who can take care of it. It is very cute and friendly but the saliva keep on drooling. overall there is nothing. forgot to mention, please keep him indoor. He likes to sleep on mattress or sofa, if can please prepare a mattress for him. PLEASE SMS , DONT CALL	2000	1.0	train
207	Unnamed	These two german shepherds are looking for a home together as their owner has passed away. One is 4 yrs old the second is 2.5-3 yrs old. The 4 yr old has a deformed back leg and recently had her tail amputated but she is able to continue walking. Good temperament. Dogs will be spayed before passing to adopter. Adoption fee will be used to cover spaying costs.	1500	NaN	test
2078	Rottweiler - Adoption	Open for Adoption with Fees Looking for new lovely home due to owner lack of time & care... Vaccination & Deworm up to date Contact me for more details	1000	2.0	train
3469	Cheras	We have just rescued a dog at 18.5km grand saga highway from Cheras heading to Kajang. We believe it is a stray and he is now undergoing a surgery at everise veterinarian clinic at Kajang Prima. According to the doctor his vertebrate t-12 and t-13 has broken and suffered from a fracture of his left tibia and fibula. The brave dog will recuperate for around one week before he can be adopted. We are currently urgently looking for people who has compassion towards injured (and may even be paral...	1000	NaN	test
4844	Coda	She is pure breed Siberian husky. Born at July . She has beautiful blue eye. Character is very gentle. Never bark or be aggressive. Her owner had to move from previous house and new place not allows dog. So we foster her for few month. I love her so much. But she requires more exercise than I can give. (Currently I walk her 1hour/day) and our	1000	2.0	train

	Name	Description	Fee	AdoptionSpeed	dataset_type
		house has no garden but only balcony. I feel that we better find suitable home (experienced dog lover with enough time and space) and current owner al...			
8834	Adpoted	adpoted	1000	0.0	train
8879	Rottweiler Semi-Adult - Adoption	Looking for new lovely home due to owner lack of time & care... Vaccination & Deworm up to date Contact me for more details	1000	1.0	train
9782	Unnamed	Available open for booking cute kitten Solid white Current just a month Ready taken end march Thank you	800	2.0	train
9745	Oscar	Oscar was found in Ara Damansara recently. My friend was walking his two dogs when this friendly chap decided to follow him home. We can't track his owner and wants to find him a good home. He has put on weight and doing well. Currently boarding at a vet. Active and playful chap, but very gentle for his size. Very affectionate and smart. Walks at yr pace without pulling the leash, knows how to sit, come up etc. The fee is not a profit _ it is to cover his boarding cost and other bills.	800	2.0	train

In [55]:
```
plt.figure(figsize=(16, 6));
plt.subplot(1, 2, 1)
plt.hist(train.loc[train['Fee'] < 400, 'Fee']);
plt.title('Distribution of fees lower than 400');

plt.subplot(1, 2, 2)
sns.violinplot(x="AdoptionSpeed", y="Fee", hue="Type", data=train);
plt.title('AdoptionSpeed by Type and Fee');
```



It's interesting to note that pets with a higher fee tend to be adopted quite rapidly. This might suggest that people prefer to pay for what they perceive as 'better' pets, such as healthy or trained animals.

The majority of pets are offered for free, with fees generally lower than $100.

Furthermore, fees for dogs tend to be higher, although these instances are relatively rare.

State

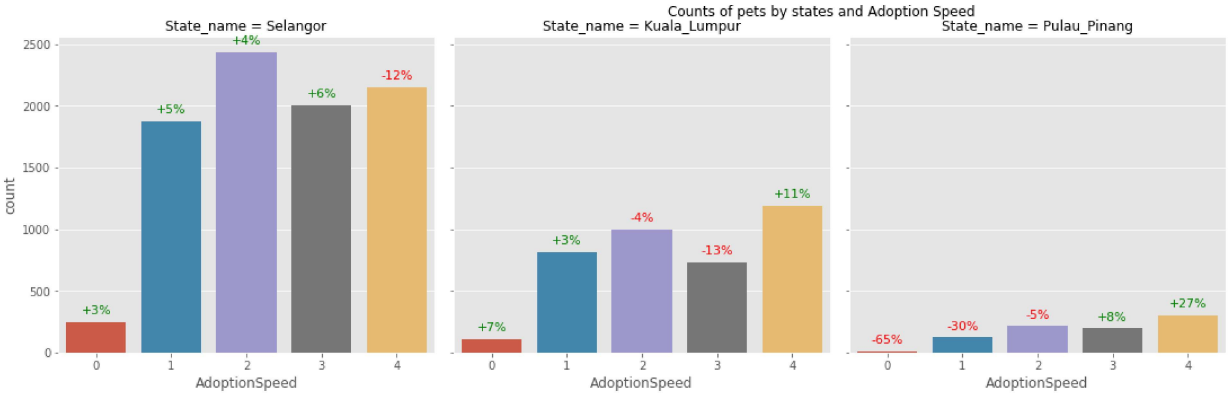
In [57]:
```
states_dict = {k: v for k, v in zip(states['StateID'], states['StateName'])}
train['State_name'] = train['State'].apply(lambda x: '_'.join(states_dict[x].split())
test['State_name'] = test['State'].apply(lambda x: '_'.join(states_dict[x].split()) if
all_data['State_name'] = all_data['State'].apply(lambda x: '_'.join(states_dict[x].spl
```

In [58]:
```
all_data['State_name'].value_counts(normalize=True).head()
```

Out[58]:
```
Selangor        0.556130
Kuala_Lumpur    0.259267
Pulau_Pinang    0.071342
Johor           0.033430
Perak           0.029897
Name: State_name, dtype: float64
```

Unfortunately, I lack information about Malaysia's states. However, I can observe that the top three states account for approximately 90% of the advertisements. Let's take a closer look at these states.

In [59]:
```
make_factor_plot(df=train.loc[train['State_name'].isin(list(train.State_name.value_cou
```



It's interesting to note that the second and third top states exhibit lower rates of adoption.

Rescuer

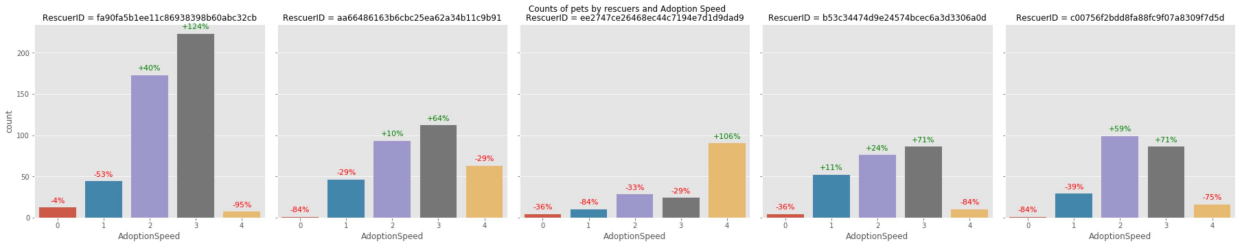
We have unique hashes for resquers.

In [60]:
```
all_data['RescuerID'].value_counts().head()
```

Out[60]:
```
fa90fa5b1ee11c86938398b60abc32cb    459
aa66486163b6cbc25ea62a34b11c9b91    315
c00756f2bdd8fa88fc9f07a8309f7d5d    231
b53c34474d9e24574bcec6a3d3306a0d    228
62a25cadb85658be5275bd54a3b8c76d    162
Name: RescuerID, dtype: int64
```

I'm curious to know whether the top five rescuers are individuals or organizations. Let's explore them to understand their nature.

In [61]:
```
make_factor_plot(df=train.loc[train['RescuerID'].isin(list(train.RescuerID.value_count
```



The rescuer with the highest number of rescued pets demonstrates the best adoption rate. Conversely, the third-ranked rescuer has the lowest adoption rate.

VideoAmt

In [62]:
```
train['VideoAmt'].value_counts()
```

Out[62]:
```
0    14419
1      417
2       92
3       36
4       15
5        7
6        4
8        2
7        1
Name: VideoAmt, dtype: int64
```

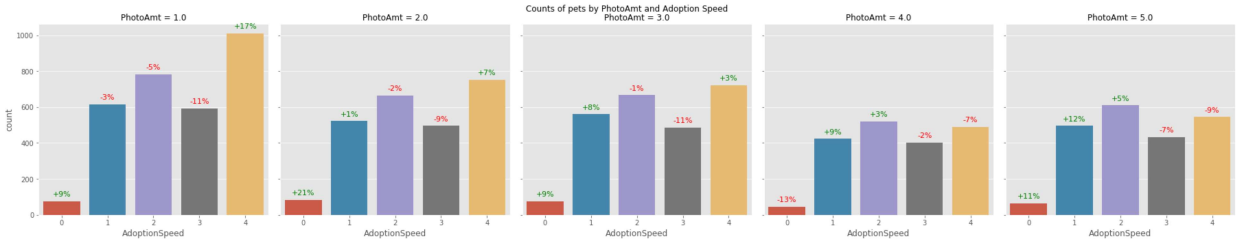
In the majority of cases, there are no videos available. Occasionally, there is one video, while having more than one video is quite rare. Due to the significant imbalance in these values, I have reservations about the potential usefulness of this variable.

PhotoAmt

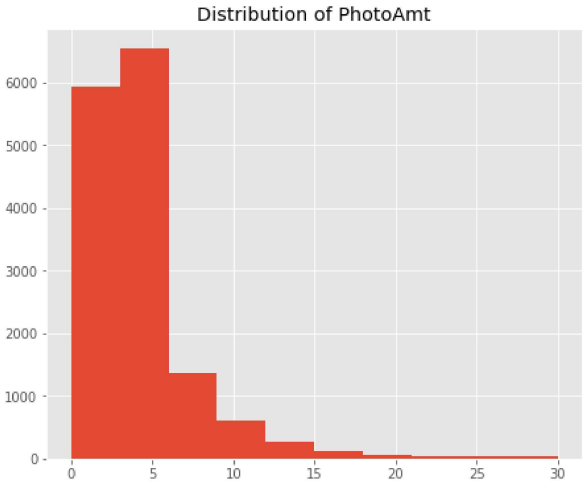
In [63]:
```
print(F'Maximum amount of photos in {train["PhotoAmt"].max()}')
train['PhotoAmt'].value_counts().head()
```

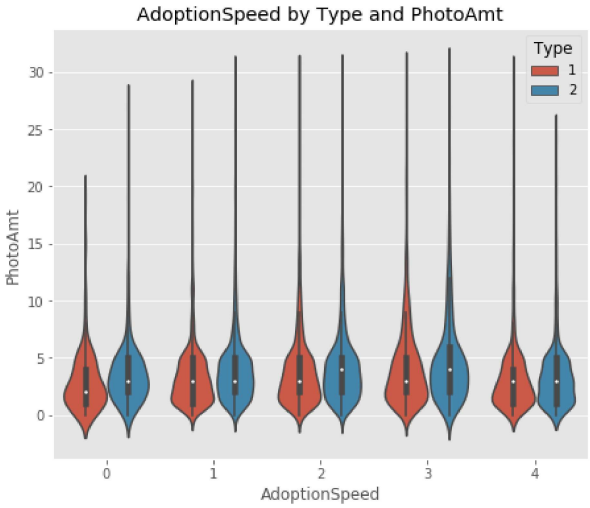
Out[63]:
```
Maximum amount of photos in 30.0
1.0    3075
2.0    2518
3.0    2511
5.0    2147
4.0    1881
Name: PhotoAmt, dtype: int64
```

In [64]:
```
make_factor_plot(df=train.loc[train['PhotoAmt'].isin(list(train.PhotoAmt.value_counts(
```



In [65]:
```
plt.figure(figsize=(16, 6));
plt.subplot(1, 2, 1)
plt.hist(train['PhotoAmt']);
plt.title('Distribution of PhotoAmt');

plt.subplot(1, 2, 2)
sns.violinplot(x="AdoptionSpeed", y="PhotoAmt", hue="Type", data=train);
plt.title('AdoptionSpeed by Type and PhotoAmt');
```

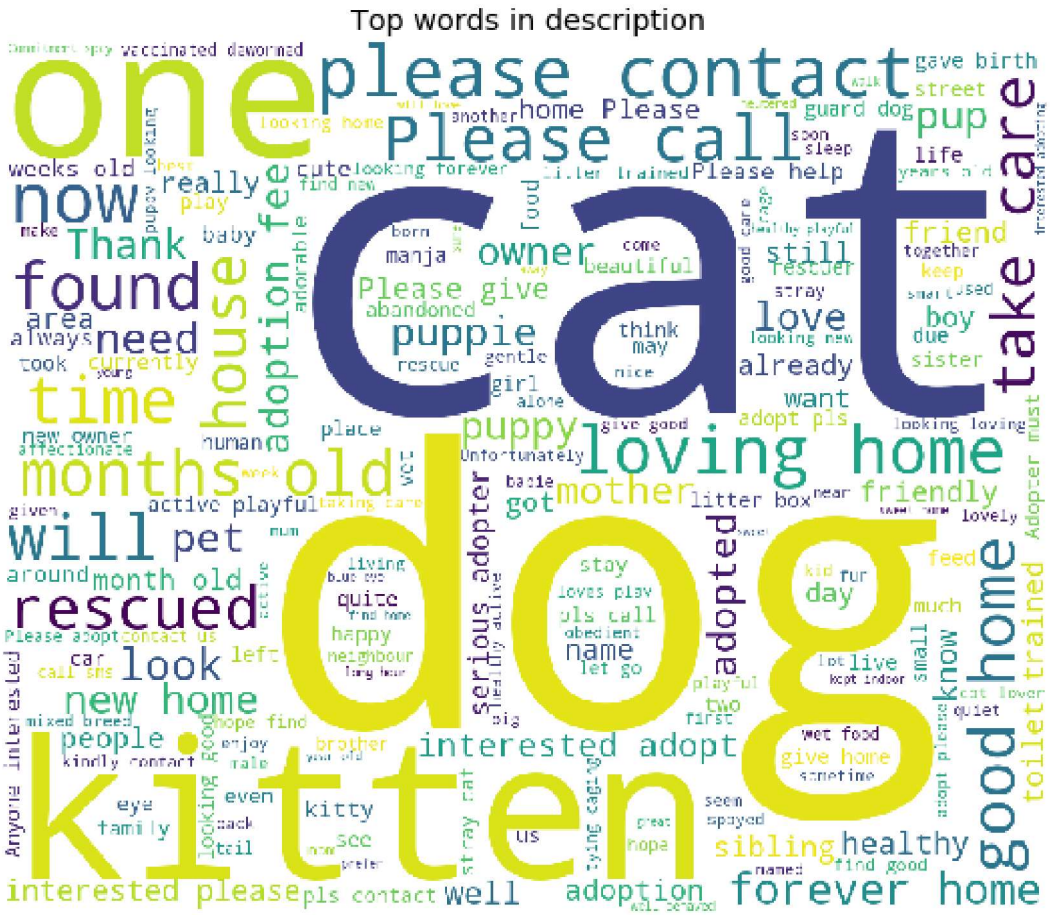




Pets can have up to 30 photos, which is a substantial amount. However, I'm skeptical about whether the quantity of photos has any significant influence.

Description

In [66]:
```
fig, ax = plt.subplots(figsize = (12, 8))
text_cat = ' '.join(all_data['Description'].fillna('').values)
wordcloud = WordCloud(max_font_size=None, background_color='white',
                      width=1200, height=1000).generate(text_cat)
plt.imshow(wordcloud)
plt.title('Top words in description');
plt.axis("off");
```



We have numerous generic words like 'cat.' To gain deeper insights, I plan to employ the ELI5 library for prediction explanation. The process involves fitting a basic vectorizer on descriptions and constructing a simple Random Forest model. This method will enable us to identify the words that influenced specific labels in the prediction.

In [67]:
```
tokenizer = TweetTokenizer()
vectorizer = TfidfVectorizer(ngram_range=(1, 2), tokenizer=tokenizer.tokenize)

vectorizer.fit(all_data['Description'].fillna('').values)
X_train = vectorizer.transform(train['Description'].fillna(''))

rf = RandomForestClassifier(n_estimators=20)
rf.fit(X_train, train['AdoptionSpeed'])
```

Out[67]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=20, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [ ]:
```
for i in range(5):
    print(f'Example of Adoption speed {i}')
    text = train.loc[train['AdoptionSpeed'] == i, 'Description'].values[0]
    print(text)
    display(eli5.show_prediction(rf, doc=text, vec=vectorizer, top=10))
```

While some words or phrases appear to be beneficial, it's becoming apparent that different adoption speed classes might share similar important words.

In [69]:
```
train['Description'] = train['Description'].fillna('')
test['Description'] = test['Description'].fillna('')
all_data['Description'] = all_data['Description'].fillna('')

train['desc_length'] = train['Description'].apply(lambda x: len(x))
train['desc_words'] = train['Description'].apply(lambda x: len(x.split()))

test['desc_length'] = test['Description'].apply(lambda x: len(x))
test['desc_words'] = test['Description'].apply(lambda x: len(x.split()))

all_data['desc_length'] = all_data['Description'].apply(lambda x: len(x))
all_data['desc_words'] = all_data['Description'].apply(lambda x: len(x.split()))

train['averate_word_length'] = train['desc_length'] / train['desc_words']
test['averate_word_length'] = test['desc_length'] / test['desc_words']
all_data['averate_word_length'] = all_data['desc_length'] / all_data['desc_words']
```

It's interesting to observe that pets with shorter ad descriptions are adopted more quickly. One possibility could be that longer descriptions might signify more issues with the pets, leading to a slower adoption speed.

Sentiment

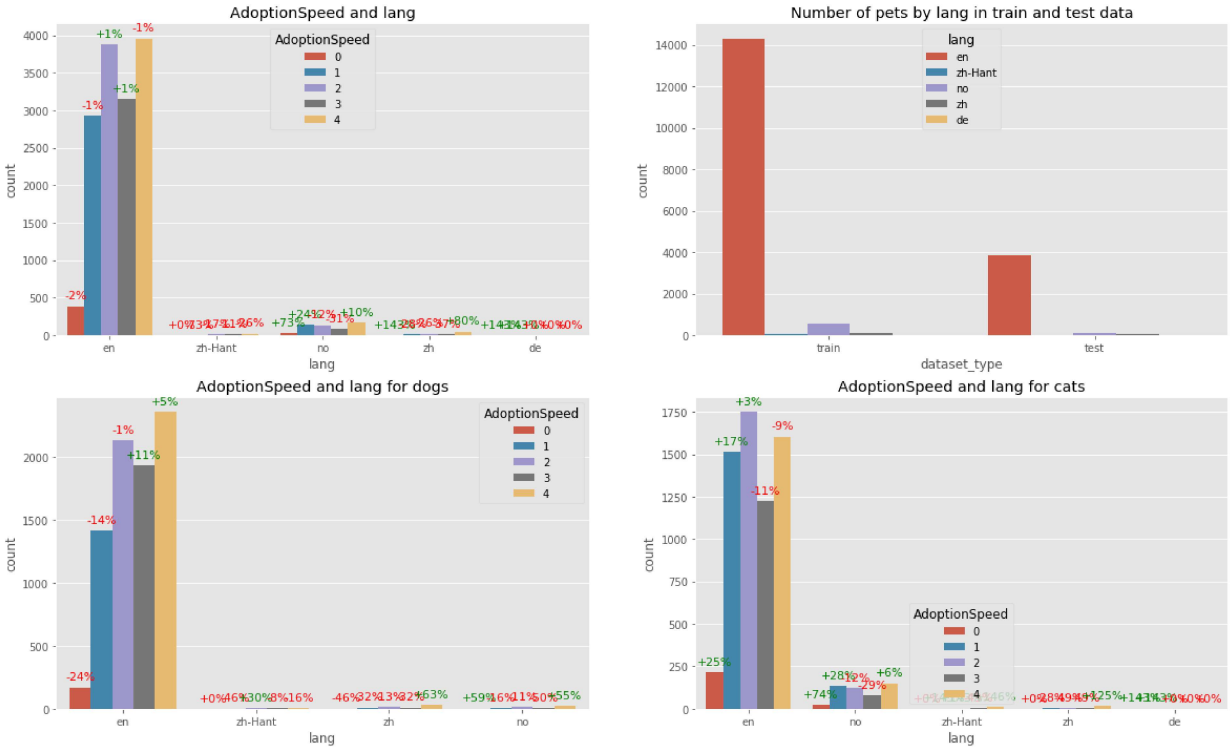
We've processed each pet profile's description through Google's Natural Language API, enabling sentiment and key entity analysis. It's important to note that there are descriptions the API couldn't analyze, resulting in fewer sentiment files than rows in the dataset.

In [71]:
```
sentiment_dict = {}
for filename in os.listdir('../input/train_sentiment/'):
    with open('../input/train_sentiment/' + filename, 'r') as f:
        sentiment = json.load(f)
    pet_id = filename.split('.')[0]
    sentiment_dict[pet_id] = {}
    sentiment_dict[pet_id]['magnitude'] = sentiment['documentSentiment']['magnitude']
    sentiment_dict[pet_id]['score'] = sentiment['documentSentiment']['score']
    sentiment_dict[pet_id]['language'] = sentiment['language']

for filename in os.listdir('../input/test_sentiment/'):
    with open('../input/test_sentiment/' + filename, 'r') as f:
        sentiment = json.load(f)
    pet_id = filename.split('.')[0]
    sentiment_dict[pet_id] = {}
    sentiment_dict[pet_id]['magnitude'] = sentiment['documentSentiment']['magnitude']
    sentiment_dict[pet_id]['score'] = sentiment['documentSentiment']['score']
    sentiment_dict[pet_id]['language'] = sentiment['language']
```

In [72]:
```
train['lang'] = train['PetID'].apply(lambda x: sentiment_dict[x]['language'] if x in s
train['magnitude'] = train['PetID'].apply(lambda x: sentiment_dict[x]['magnitude'] if
train['score'] = train['PetID'].apply(lambda x: sentiment_dict[x]['score'] if x in ser

test['lang'] = test['PetID'].apply(lambda x: sentiment_dict[x]['language'] if x in ser
```

```
test['magnitude'] = test['PetID'].apply(lambda x: sentiment_dict[x]['magnitude'] if x
test['score'] = test['PetID'].apply(lambda x: sentiment_dict[x]['score'] if x in senti

all_data['lang'] = all_data['PetID'].apply(lambda x: sentiment_dict[x]['language'] if
all_data['magnitude'] = all_data['PetID'].apply(lambda x: sentiment_dict[x]['magnitude
all_data['score'] = all_data['PetID'].apply(lambda x: sentiment_dict[x]['score'] if x
```

In [73]:
```
plot_four_graphs(col='lang', main_title='lang', dataset_title='Number of pets by lang
```



It seems that the lower is the magnitude of score, the faster pets are adopted.

Basic model

In [75]:
```
cols_to_use = ['Type', 'Age', 'Breed1', 'Breed2', 'Gender', 'Color1', 'Color2',
       'Color3', 'MaturitySize', 'FurLength', 'Vaccinated', 'Dewormed',
       'Sterilized', 'Health', 'Quantity', 'Fee', 'State', 'RescuerID', 'health', 'Fre
       'VideoAmt', 'PhotoAmt', 'AdoptionSpeed', 'No_name', 'Pure_breed', 'desc_length'
train = train[[col for col in cols_to_use if col in train.columns]]
test = test[[col for col in cols_to_use if col in test.columns]]
```

In [76]:
```
cat_cols = ['Type', 'Breed1', 'Breed2', 'Gender', 'Color1', 'Color2',
       'Color3', 'MaturitySize', 'FurLength', 'Vaccinated', 'Dewormed',
       'Sterilized', 'Health', 'State', 'RescuerID',
       'No_name', 'Pure_breed', 'health', 'Free']
```

In [77]:
```
more_cols = []
for col1 in cat_cols:
    for col2 in cat_cols:
        if col1 != col2 and col1 not in ['RescuerID', 'State'] and col2 not in ['Rescu
            train[col1 + '_' + col2] = train[col1].astype(str) + '_' + train[col2].ast
            test[col1 + '_' + col2] = test[col1].astype(str) + '_' + test[col2].astype
            more_cols.append(col1 + '_' + col2)
```

```
cat_cols = cat_cols + more_cols
```

In [78]:
```
%%time
indexer = {}
for col in cat_cols:
    # print(col)
    _, indexer[col] = pd.factorize(train[col].astype(str))

for col in tqdm_notebook(cat_cols):
    # print(col)
    train[col] = indexer[col].get_indexer(train[col].astype(str))
    test[col] = indexer[col].get_indexer(test[col].astype(str))
```

```
HBox(children=(IntProgress(value=0, max=291), HTML(value='')))
CPU times: user 8.19 s, sys: 81.4 ms, total: 8.28 s
Wall time: 8.22 s
```

In [79]:
```
y = train['AdoptionSpeed']
train = train.drop(['AdoptionSpeed'], axis=1)
```

Naive multiclass LGB

In [80]:
```
n_fold = 5
folds = StratifiedKFold(n_splits=n_fold, shuffle=True, random_state=15)
```

In [81]:
```
def train_model(X=train, X_test=test, y=y, params=None, folds=folds, model_type='lgb',
    result_dict = {}
    if make_oof:
        oof = np.zeros((len(X), 5))
    prediction = np.zeros((len(X_test), 5))
    scores = []
    feature_importance = pd.DataFrame()
    for fold_n, (train_index, valid_index) in enumerate(folds.split(X, y)):
        gc.collect()
        print('Fold', fold_n + 1, 'started at', time.ctime())
        X_train, X_valid = X.iloc[train_index], X.iloc[valid_index]
        y_train, y_valid = y.iloc[train_index], y.iloc[valid_index]


        if model_type == 'lgb':
            train_data = lgb.Dataset(X_train, label=y_train, categorical_feature = cat
            valid_data = lgb.Dataset(X_valid, label=y_valid, categorical_feature = cat

            model = lgb.train(params,
                    train_data,
                    num_boost_round=20000,
                    valid_sets = [train_data, valid_data],
                    verbose_eval=500,
                    early_stopping_rounds = 200)

            del train_data, valid_data

            y_pred_valid = model.predict(X_valid, num_iteration=model.best_iteration)
            del X_valid
            gc.collect()
            y_pred = model.predict(X_test, num_iteration=model.best_iteration)
```

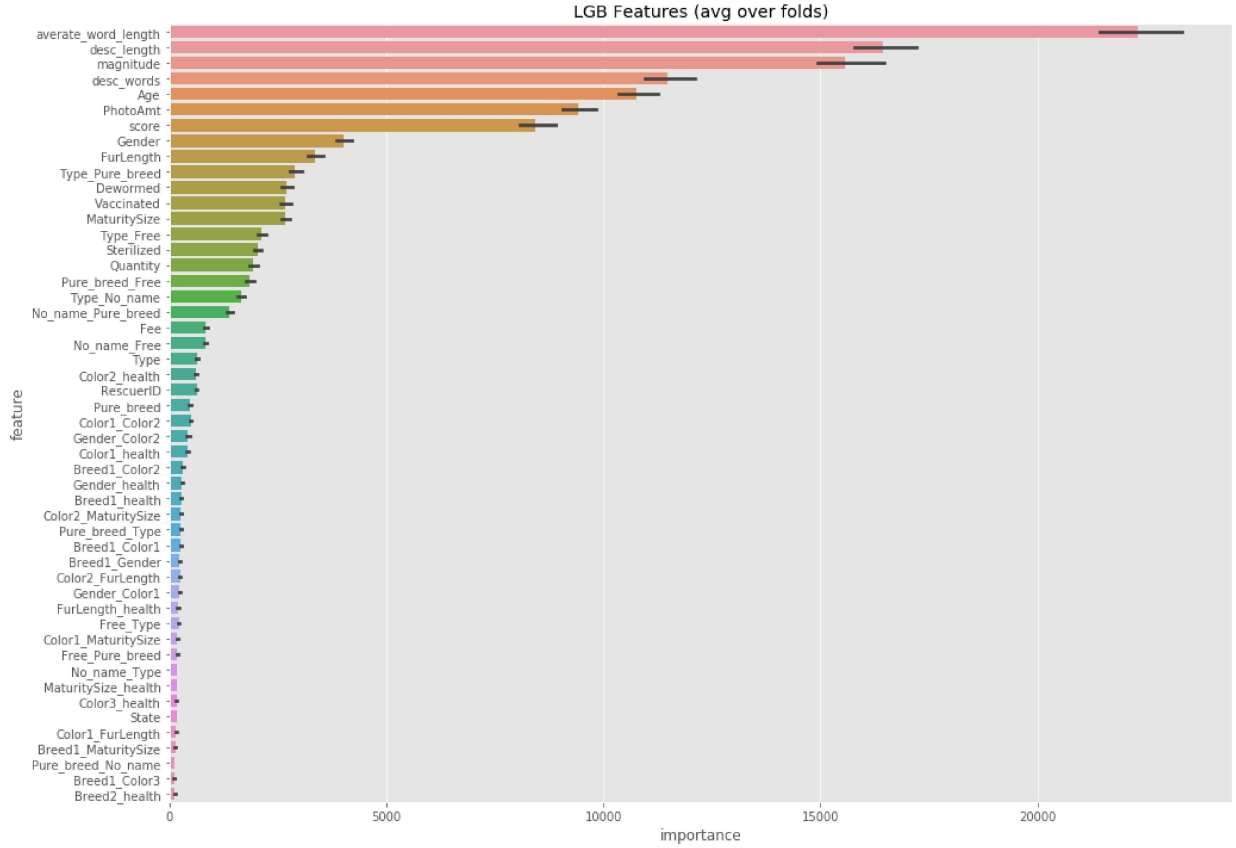
```
        if model_type == 'xgb':
            train_data = xgb.DMatrix(data=X_train, label=y_train)
            valid_data = xgb.DMatrix(data=X_valid, label=y_valid)

            watchlist = [(train_data, 'train'), (valid_data, 'valid_data')]
            model = xgb.train(dtrain=train_data, num_boost_round=20000, evals=watchlis
            y_pred_valid = model.predict(xgb.DMatrix(X_valid), ntree_limit=model.best_
            y_pred = model.predict(xgb.DMatrix(X_test), ntree_limit=model.best_ntree_l

        if model_type == 'lcv':
            model = LogisticRegressionCV(scoring='neg_log_loss', cv=3, multi_class='mu
            model.fit(X_train, y_train)

            y_pred_valid = model.predict(X_valid)
            y_pred = model.predict(X_test)

        if model_type == 'cat':
            model = CatBoostClassifier(iterations=20000,  loss_function='MultiClass',
            model.fit(X_train, y_train, eval_set=(X_valid, y_valid), cat_features=[],

            y_pred_valid = model.predict(X_valid)
            y_pred = model.predict(X_test).reshape(-1,)

        if make_oof:
            oof[valid_index] = y_pred_valid

        scores.append(kappa(y_valid, y_pred_valid.argmax(1)))
        print('Fold kappa:', kappa(y_valid, y_pred_valid.argmax(1)))
        print('')

        if averaging == 'usual':
            prediction += y_pred
        elif averaging == 'rank':
            prediction += pd.Series(y_pred).rank().values

        if model_type == 'lgb':
            # feature importance
            fold_importance = pd.DataFrame()
            fold_importance["feature"] = X.columns
            fold_importance["importance"] = model.feature_importance()
            fold_importance["fold"] = fold_n + 1
            feature_importance = pd.concat([feature_importance, fold_importance], axis

    prediction /= n_fold

    print('CV mean score: {0:.4f}, std: {1:.4f}.'.format(np.mean(scores), np.std(score

    if model_type == 'lgb':

        if plot_feature_importance:
            feature_importance["importance"] /= n_fold
            cols = feature_importance[["feature", "importance"]].groupby("feature").me
                by="importance", ascending=False)[:50].index

            best_features = feature_importance.loc[feature_importance.feature.isin(col

            plt.figure(figsize=(16, 12));
            sns.barplot(x="importance", y="feature", data=best_features.sort_values(by
            plt.title('LGB Features (avg over folds)');
```

```
            result_dict['feature_importance'] = feature_importance

    result_dict['prediction'] = prediction
    if make_oof:
        result_dict['oof'] = oof

    return result_dict
```

In [82]:
```
params = {'num_leaves': 512,
        #  'min_data_in_leaf': 60,
         'objective': 'multiclass',
         'max_depth': -1,
         'learning_rate': 0.01,
         "boosting": "gbdt",
         "feature_fraction": 0.9,
         "bagging_freq": 3,
         "bagging_fraction": 0.9,
         "bagging_seed": 11,
        #  "lambda_l1": 0.1,
         # "lambda_l2": 0.1,
         "random_state": 42,
         "verbosity": -1,
         "num_class": 5}
```

In [83]:
```
result_dict_lgb = train_model(X=train, X_test=test, y=y, params=params, model_type='lg
```

```
Fold 1 started at Mon Oct 23 06:01:31 2023
Training until validation scores don't improve for 200 rounds.
[500]   training's multi_logloss: 0.364013      valid_1's multi_logloss: 1.31193
Early stopping, best iteration is:
[315]   training's multi_logloss: 0.581701      valid_1's multi_logloss: 1.29846
Fold kappa: 0.3603590469022747

Fold 2 started at Mon Oct 23 06:03:02 2023
Training until validation scores don't improve for 200 rounds.
[500]   training's multi_logloss: 0.363189      valid_1's multi_logloss: 1.30985
Early stopping, best iteration is:
[321]   training's multi_logloss: 0.572464      valid_1's multi_logloss: 1.30093
Fold kappa: 0.33643133750372367

Fold 3 started at Mon Oct 23 06:04:32 2023
Training until validation scores don't improve for 200 rounds.
[500]   training's multi_logloss: 0.365049      valid_1's multi_logloss: 1.29835
Early stopping, best iteration is:
[365]   training's multi_logloss: 0.512845      valid_1's multi_logloss: 1.29124
Fold kappa: 0.3462602014645225

Fold 4 started at Mon Oct 23 06:06:11 2023
Training until validation scores don't improve for 200 rounds.
[500]   training's multi_logloss: 0.363249      valid_1's multi_logloss: 1.30607
Early stopping, best iteration is:
[332]   training's multi_logloss: 0.555928      valid_1's multi_logloss: 1.29644
Fold kappa: 0.3803747246864204

Fold 5 started at Mon Oct 23 06:07:43 2023
Training until validation scores don't improve for 200 rounds.
[500]   training's multi_logloss: 0.364018      valid_1's multi_logloss: 1.29869
Early stopping, best iteration is:
[337]   training's multi_logloss: 0.550079      valid_1's multi_logloss: 1.28968
Fold kappa: 0.4100616366773082

CV mean score: 0.3667, std: 0.0262.
```



In [84]:
```
xgb_params = {'eta': 0.01, 'max_depth': 9, 'subsample': 0.9, 'colsample_bytree': 0.9,
              'objective': 'multi:softprob', 'eval_metric': 'merror', 'silent': True, 'nth
result_dict_xgb = train_model(params=xgb_params, model_type='xgb', make_oof=True)
```

```
Fold 1 started at Mon Oct 23 06:09:18 2023
[0]     train-merror:0.443759   valid_data-merror:0.632
Multiple eval metrics have been passed: 'valid_data-merror' will be used for early st
opping.

Will train until valid_data-merror hasn't improved in 200 rounds.
[500]   train-merror:0.107396   valid_data-merror:0.557
Stopping. Best iteration:
[413]   train-merror:0.12574    valid_data-merror:0.554

Fold kappa: 0.3710262949400668

Fold 2 started at Mon Oct 23 06:14:01 2023
[0]     train-merror:0.432919   valid_data-merror:0.617333
Multiple eval metrics have been passed: 'valid_data-merror' will be used for early st
opping.

Will train until valid_data-merror hasn't improved in 200 rounds.
[500]   train-merror:0.106312   valid_data-merror:0.567333
Stopping. Best iteration:
[631]   train-merror:0.084966   valid_data-merror:0.561667

Fold kappa: 0.3475632365506456

Fold 3 started at Mon Oct 23 06:20:20 2023
[0]     train-merror:0.424844   valid_data-merror:0.639426
Multiple eval metrics have been passed: 'valid_data-merror' will be used for early st
opping.

Will train until valid_data-merror hasn't improved in 200 rounds.
[500]   train-merror:0.11163    valid_data-merror:0.581054
Stopping. Best iteration:
[402]   train-merror:0.131055   valid_data-merror:0.577385

Fold kappa: 0.3399364661585771

Fold 4 started at Mon Oct 23 06:24:59 2023
[0]     train-merror:0.433097   valid_data-merror:0.613742
Multiple eval metrics have been passed: 'valid_data-merror' will be used for early st
opping.

Will train until valid_data-merror hasn't improved in 200 rounds.
[500]   train-merror:0.104043   valid_data-merror:0.56004
Stopping. Best iteration:
[366]   train-merror:0.134389   valid_data-merror:0.557705

Fold kappa: 0.38481752708779915

Fold 5 started at Mon Oct 23 06:29:21 2023
[0]     train-merror:0.443398   valid_data-merror:0.627961
Multiple eval metrics have been passed: 'valid_data-merror' will be used for early st
opping.

Will train until valid_data-merror hasn't improved in 200 rounds.
[500]   train-merror:0.10095    valid_data-merror:0.561895
Stopping. Best iteration:
[424]   train-merror:0.115205   valid_data-merror:0.558225

Fold kappa: 0.4029977159117136
```

```
CV mean score: 0.3693, std: 0.0233.
```

In [85]:
```
prediction = (result_dict_lgb['prediction'] + result_dict_xgb['prediction']).argmax(1)
Result = pd.DataFrame({'PetID': sub.PetID, 'AdoptionSpeed': [int(i) for i in predictic
Result.head()
```

Out[85]:

	PetID	AdoptionSpeed
0	e2dfc2935	2
1	f153b465f	2
2	3c90f3f54	2
3	e02abc8a3	4
4	09f0df7d1	3

In [86]:
```
Result.to_csv('Result.csv', index=False)
```