DATA BOOTCAMP PROJECT - PET ADOPTION

OVERVIEW

Data Scientists for this project: Shimeng Cao (sc6755@stern.nyu.edu), Jiarong Li (jl9175@stern.nyu.edu), and Evan Okin (eo919@stern.nyu.edu)

For our project, we analyzed pet adoption data for cats and dogs. We scoured data science competitions on Kaggle.com to look for a dataset that we were passionate about. We found a competition called "Pet Adoption Speed Prediction" which provides roughly 15,000 rows data for cats and dogs. As people who have adopted pets in the past, we decided that this would be a fun project. In a best case scenario, we might even find enough insights to share with a pet adoption agency to improve the likelihood of pets being adopted.

To help readers better follow our project, we structured our analysis in below flow：

- Data Exploration and Cleaning
- Regression Analysis
- Machine Learning Algorithms: 1) ML for the regression problem 2) ML for the classification problem

IMPORT PACKAGES

We start by importing necessary packages and libraries. This will allow us to read in our data, perform operations more fluently on dataframes, and create some beautiful visualizations. Additionally, we will be able to perform regressions and create prediction models using machine learning algorithms.

In [68]:
```
#Working With DataFrames
import pandas as pd
import numpy as np
#Data Visualizations
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
#Regressions and Machine Learning
import statsmodels.formula.api as smf
import patsy
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression as reg
from sklearn.linear_model import LogisticRegression as logistic
from sklearn.neighbors import KNeighborsRegressor as knn
from sklearn.ensemble import RandomForestRegressor as rf
from sklearn.neighbors import KNeighborsClassifier as knnc
from sklearn.ensemble import RandomForestClassifier as rfc
from sklearn.model_selection import train_test_split
```

We saved the data from Kaggle.com Competiton (Pet Adoption Speed Prediction) to our desktops (where we set our working directory to be via terminal, where our jupyter notebook is also set up). Next, we read in the dataset, which we saved as a csv (this is the convention file type for large datasets).

In [4]:
```
df=pd.read_csv('/Users/jessica.li/Desktop/dataset_final_project.csv')
```

DATA EXPLORATION AND CLEANING

The following are the data fields corresponding to our project, which we obtained on Kaggle.com:

PetID - Unique hash ID of pet profile
AdoptionSpeed - Categorical speed of adoption. Lower is faster. This is the value to predict. See below section for more info.
Type - Type of animal (1 = Dog, 2 = Cat)
Name - Name of pet (Empty if not named)
Age - Age of pet when listed, in months
Breed1 - Primary breed of pet (Refer to BreedLabels dictionary)
Breed2 - Secondary breed of pet, if pet is of mixed breed (Refer to BreedLabels dictionary)
Gender - Gender of pet (1 = Male, 2 = Female, 3 = Mixed, if profile represents group of pets)
Color1 - Color 1 of pet (Refer to ColorLabels dictionary)
Color2 - Color 2 of pet (Refer to ColorLabels dictionary)
Color3 - Color 3 of pet (Refer to ColorLabels dictionary)
MaturitySize - Size at maturity (1 = Small, 2 = Medium, 3 = Large, 4 = Extra Large, 0 = Not Specified)
FurLength - Fur length (1 = Short, 2 = Medium, 3 = Long, 0 = Not Specified)
Vaccinated - Pet has been vaccinated (1 = Yes, 2 = No, 3 = Not Sure)
Dewormed - Pet has been dewormed (1 = Yes, 2 = No, 3 = Not Sure)
Sterilized - Pet has been spayed / neutered (1 = Yes, 2 = No, 3 = Not Sure)
Health - Health Condition (1 = Healthy, 2 = Minor Injury, 3 = Serious Injury, 0 = Not Specified)
Quantity - Number of pets represented in profile
Fee - Adoption fee (0 = Free)
State - State location in Malaysia (Refer to StateLabels dictionary)
RescuerID - Unique hash ID of rescuer
VideoAmt - Total uploaded videos for this pet
PhotoAmt - Total uploaded photos for this pet
Description - Profile write-up for this pet. The primary language used is English, with some in Malay or Chinese.

We wanted to explore the data. Ultimately, we wanted to figure out the predictive nature of these variables in determining whether or not we could predict (one way or another) whether a pet would be adopted. At first glance, we saw a variable PetID, which we believed should have no predictive value. On the other hand, the variable Vaccinated is likely to be predictive (before analyzing the data, we believed that a pet that is vaccinated would be more likely to be adopted than a pet that is not vaccinated).

We looked at the first three rows of the dataframe, to make sure that it loaded into python properly.

In [5]:
```
df.head(3)
```

Out[5]:

	Type	Name	Age	Breed1	Breed2	Gender	Color1	Color2	Color3	MaturitySize	...	Health
0	2.0	Nibble	3	299	0	1.0	1	7	0	1	...	1
1	2.0	No Name Yet	1	265	0	1.0	1	2	0	2	...	1
2	1.0	Brisco	1	307	0	1.0	2	7	0	2	...	1

3 rows × 24 columns

We wanted to see how many rows and columns the dataset has.

In [6]:
```
df.shape
```

Out[6]:
```
(14993, 24)
```

The dataset has 14,993 rows and 24 columns. With 24 columns, we have a lot of potential variables to aid in our prediction models.

We wanted to explore the variables.

In [7]:
```
df.describe()
```

Out[7]:

	Type	Age	Breed1	Breed2	Gender	Color1	
count	14991.000000	14993.000000	14993.000000	14993.000000	14991.000000	14993.000000	149!
mean	1.457608	10.452078	265.272594	74.009738	1.776132	2.234176	
std	0.498216	18.155790	60.056818	123.011575	0.681535	1.745225	
min	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000	
25%	1.000000	2.000000	265.000000	0.000000	1.000000	1.000000	
50%	1.000000	3.000000	266.000000	0.000000	2.000000	2.000000	
75%	2.000000	12.000000	307.000000	179.000000	2.000000	3.000000	
max	2.000000	255.000000	307.000000	307.000000	3.000000	7.000000	

In class, we learned that convention is to use lower-case for columns in a dataframe. We clean up the column names per convention before diving into deeper analysis. Consistent headers will make it easier to run regressions as well.

In [8]:
```
df.columns=[i.lower() for i in df.columns]
```

Next, we found the amount of null values by column. Null values of a column could be a bad thing - if we find a null value for a column that we care about, we might have to throw out the entire row/datapoint. On the other hand, if the null value is for a variable that is not predictive, it might not matter. In some cases, it will be fine to fill in the data with key statistics such as the mean or mode. Either way, it's important to have a clean dataset.

In [9]:
```
df.isna().sum()
```

Out[9]:
```
type                2
name             1257
age                 0
breed1              0
breed2              0
gender              2
color1              0
color2              0
color3              0
maturitysize        0
furlength           0
vaccinated          0
dewormed            0
sterilized          0
health              0
quantity            0
fee                 1
state               0
rescuerid           0
videoamt            0
description        12
petid               0
photoamt            0
adoptionspeed       0
dtype: int64
```

First, we dealt with "type." This variable tells us if the pet is a dog or a cat. With missing type, we don't know if the data point is a dog or a cat, and therefor we dropped the entire row.

In [10]:
```
df = df.dropna(subset=['type'])
```

In [11]:
```
df.isna().sum()
```

Out[11]:
```
type                0
name             1257
age                 0
breed1              0
breed2              0
gender              2
color1              0
color2              0
color3              0
maturitysize        0
furlength           0
vaccinated          0
dewormed            0
sterilized          0
health              0
quantity            0
fee                 1
state               0
rescuerid           0
videoamt            0
description        12
petid               0
photoamt            0
adoptionspeed       0
dtype: int64
```

Next we dealt with the null value for "fee." We looked at the distribution:

In [12]:
```
df['fee'].value_counts().head(5)/len(df['fee'])
```

Out[12]:
```
0.0      0.844573
50.0     0.031219
100.0    0.027216
200.0    0.014609
150.0    0.010806
Name: fee, dtype: float64
```

Nearly 85% of the fee values are 0, so we decided to replace null values with the (overwhelmingly popular) mode.

In [13]:
```
df['fee'].fillna(0,inplace=True)
```

In [14]:
```
df.isna().sum()
```

Out[14]:
```
type               0
name            1257
age                0
breed1             0
breed2             0
gender             2
color1             0
color2             0
color3             0
maturitysize       0
furlength          0
vaccinated         0
dewormed           0
sterilized         0
health             0
quantity           0
fee                0
state              0
rescuerid          0
videoamt           0
description       12
petid              0
photoamt           0
adoptionspeed      0
dtype: int64
```

We decided to remove the variable "name" since it likely has no predictive power and the column has a lot of missing values.

In [15]:
```
df = df.drop('name',axis=1).dropna()
df.shape
```

Out[15]:
```
(14977, 23)
```

In [16]:
```
df.isna().sum()
```

Out[16]:
```
type             0
age              0
breed1           0
breed2           0
gender           0
color1           0
color2           0
color3           0
maturitysize     0
furlength        0
vaccinated       0
dewormed         0
sterilized       0
health           0
quantity         0
fee              0
state            0
rescuerid        0
videoamt         0
description      0
petid            0
photoamt         0
adoptionspeed    0
dtype: int64
```

According to the description, type 1 is a dog and type 2 is a cat. But, we want to see if we could figure that out on our own, by analyzing the description string.

In [17]:
```
df_type_test=df.loc[df['description'].str.contains('dog'),:]
df_type_test['type'].value_counts()/len(df_type_test)
```

Out[17]:
```
1.0    0.924496
2.0    0.075504
Name: type, dtype: float64
```

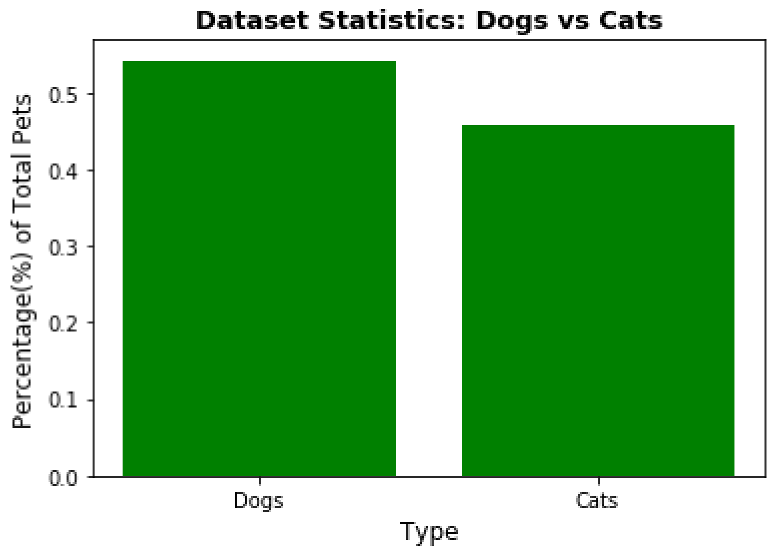
Over 92% of the description strings that include the string"dog" are type 1! Thus, we would also have concluded that type 1 is a dog.

We found the percentage of the dataset that is dogs vs. cats, and began visualizing.

In [18]:
```
df['type'].value_counts()/len(df['type'])
```

Out[18]:
```
1.0    0.542632
2.0    0.457368
Name: type, dtype: float64
```

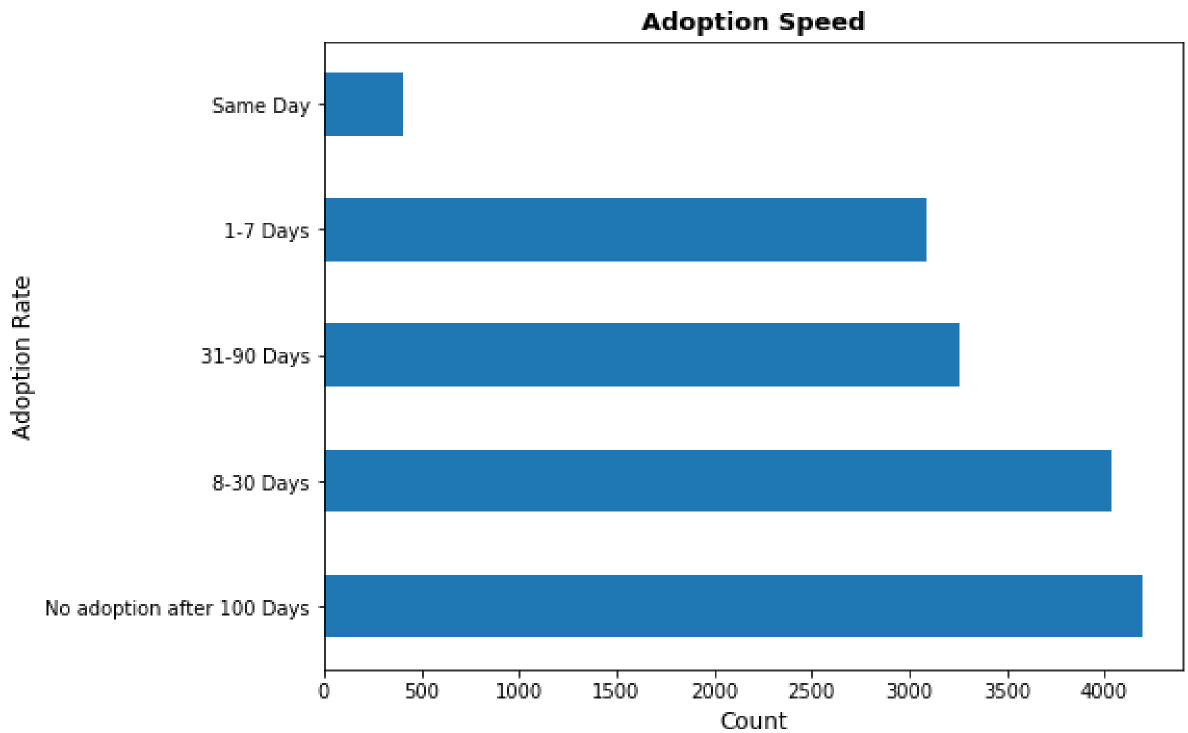
In [19]:
```
x=['Dogs','Cats']
y=[.542632,.457368]
plt.bar(x,y,color='g')
plt.title('Dataset Statistics: Dogs vs Cats',fontsize=13,fontweight='bol
d')
plt.xlabel('Type',fontsize=12)
plt.ylabel('Percentage(%) of Total Pets',fontsize=12)
plt.show()
```



The sad truth is that many pets don't get adopted quickly (or, they don't get adopted at all). We then analyzed pet adoption speed. Since our last plot was a vertical bar chart, we used a horizontal bar chart.

In [20]:
```
df['adoptionspeed'].value_counts().rename(
    {0:'Same Day',
     1:'1-7 Days',
     2:'8-30 Days',
     3:'31-90 Days',
     4:'No adoption after 100 Days'}).plot(kind='barh',figsize=(8,6))
plt.title('Adoption Speed', fontsize=13,fontweight='bold')
plt.xlabel('Count',fontsize=12)
plt.ylabel('Adoption Rate',fontsize=12)
```

Out[20]:
```
Text(0, 0.5, 'Adoption Rate')
```



We analyzed counts by adoption speed, and found the percentage of pets that have not been adopted after 100 days.

In [21]:
```
df['adoptionspeed'].value_counts()
```

Out[21]:
```
4    4193
2    4031
3    3255
1    3088
0     410
Name: adoptionspeed, dtype: int64
```

In [22]:
```
round(len(df.loc[df['adoptionspeed']==4])/len(df),2)
```

Out[22]:
```
0.28
```

Sadly, nearly 28% of all pets in our dataset were not adopted after 100 days.

We wanted to find out, is the adoption speed better (with a corresponding smaller value) for dogs or cats?

In [23]:
```
df.groupby(['type']).agg({'adoptionspeed':'mean'})
```

Out[23]:

type	adoptionspeed
1.0	2.615233
2.0	2.398978

This data shows us that cats are more likely to be adopted quicker than dogs.

We found that 28% of all pets are not adopted after 100 days. We decided to filter on several characteristics to see if we could deduce some characteristics which help make pets more likely to be adopted. We filtered on pets who possess all of the following characteristics at the same time - are vaccinated, are dewormed, are sterilized, are either healthy or have only a minor injury, have no adoption fee, and age of less than 1 year.

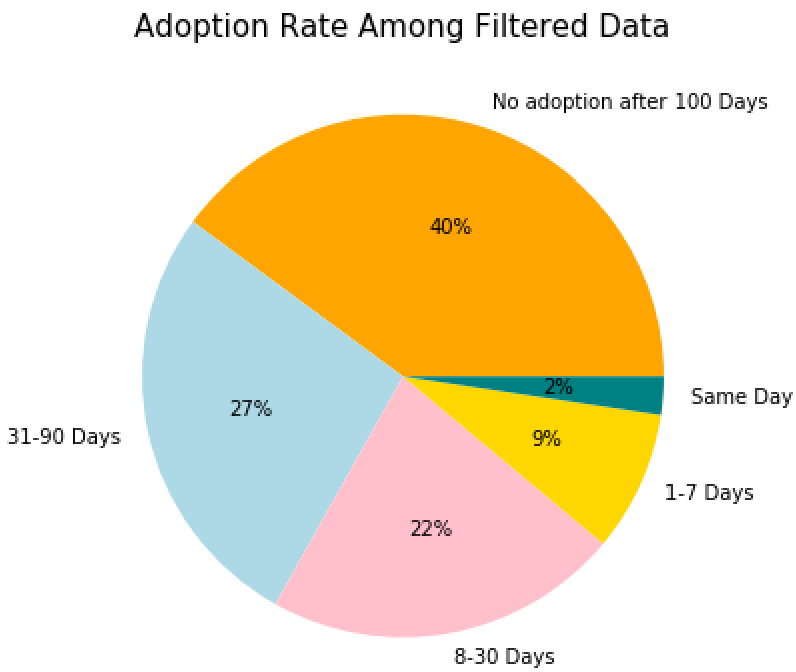
In [24]:
```
vaccinated_list=[1]
dewormed_list=[1]
sterilized_list=[1]
health_list=[1,2]
fee_list=[0]
age_list=[1,2,3,4,5,6,7,8,9,10,11,12]

df_filter_adj=df.loc[df['vaccinated'].isin(vaccinated_list)&df['deworme
d'].isin(dewormed_list)&
        df['sterilized'].isin(sterilized_list)&df['health'].isin(health_l
ist)&
        df['fee'].isin(fee_list)&df['age'].isin(age_list)]
df_filter_adj['adoptionspeed'].value_counts()/len(df_filter_adj)
```

Out[24]:
```
4    0.398974
3    0.269744
2    0.220513
1    0.087179
0    0.023590
Name: adoptionspeed, dtype: float64
```

We looked at this in a pie-chart.

In [25]:
```
labels=['No adoption after 100 Days','31-90 Days','8-30 Days','1-7 Days'
,'Same Day']
sizes=df_filter_adj['adoptionspeed'].value_counts().values
colors=['orange','lightblue','pink','gold','teal']
explode=[0,0,0,0,0]
plt.style.use('seaborn-pastel')
plt.figure(figsize=(6,6))
plt.pie(sizes,explode=explode,labels=labels,colors=colors,autopct= '%1.0
f%%')
plt.title('Adoption Rate Among Filtered Data',fontsize=15)
plt.show()
labels
```

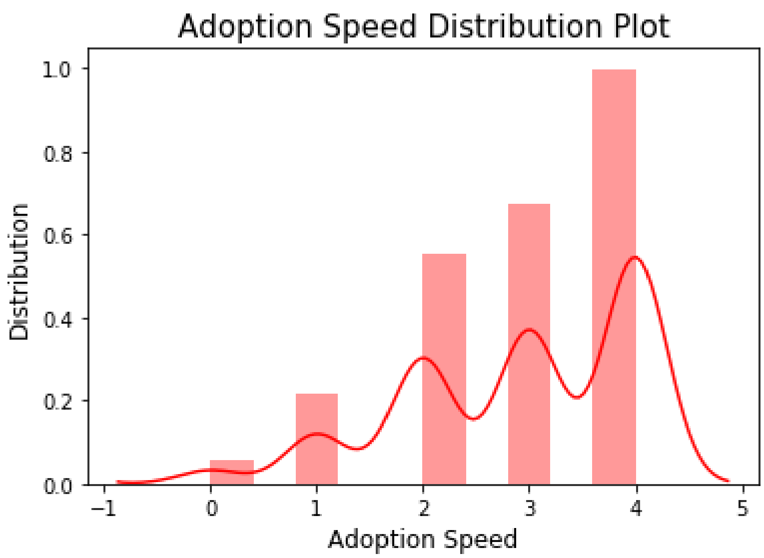


Out[25]:
```
['No adoption after 100 Days',
 '31-90 Days',
 '8-30 Days',
 '1-7 Days',
 'Same Day']
```

Or, as a plot.

In [26]:
```
sns.distplot(df_filter_adj['adoptionspeed'],color = "red")
plt.title('Adoption Speed Distribution Plot',fontsize=15)
plt.xlabel('Adoption Speed',fontsize=12)
plt.ylabel('Distribution',fontsize=12)
```

Out[26]: Text(0, 0.5, 'Distribution')



Surprisingly, 40% of our sliced dataset were not adopted. However, it's possible that with less than 1,000 datapoints for this new dataframe, there's just not enough information for us to draw a conclusion one way or the other. Also, it's possible that the adoption agency does everything they can to make an undesirable adoption pet seem more desirable (such as giving vaccinations).

One of our hypotheses was that a longer description length means that a pet is more likely to be adopted. We created a new column in our dataframe with the character length of the description, and then analyzed that column.

In [27]:
```
df['description_length']=df['description'].str.len()
```

In [28]:
```
np.corrcoef(df['description_length'],-df['adoptionspeed'])[0,1]
```

Out[28]: 0.010236029488048715

There is some positive correlation (approximately 1%), but not enough to be labeled significant. The reason that we took the negative of the column for adoption speed is that a smaller value is "better" than a larger value.

We believed there would be a positive relationship/correlation between whether a pet has been vaccinated and adoption speed.

In [29]:
```
np.corrcoef(df['vaccinated'],-df['adoptionspeed'])[0,1]
```

Out[29]: 0.059697201203509416

As expected, there is some correlation, although it's smaller than we expected.

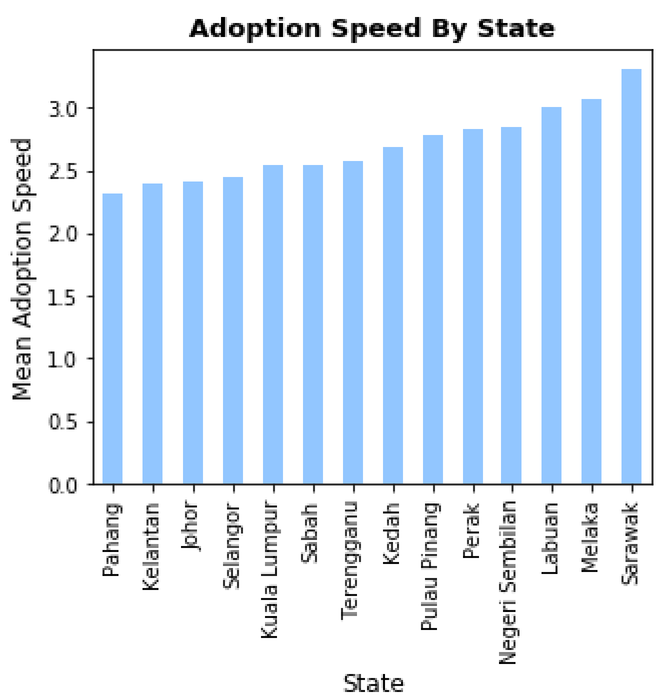
We learned in class how to merge two datasets together. We wanted to merge state name to the original dataframe.

In [31]:
```
state=pd.read_csv('/Users/jessica.li/Desktop/state_labels.csv')
state = state.rename(columns={'StateID':'state','StateName':'state_name'
})
state
df = df.merge(state,
        on='state',
        how = 'left')
```

With the merged dataframe, we could assess adoption speed by state, using groupby.

In [32]:
```
df.groupby('state_name')['adoptionspeed'].mean().sort_values().plot.bar(
figsize=(5,4))
plt.title('Adoption Speed By State',fontsize=13,fontweight='bold')
plt.xlabel('State',fontsize=12)
plt.ylabel('Mean Adoption Speed',fontsize=12)
```

Out[32]: Text(0, 0.5, 'Mean Adoption Speed')

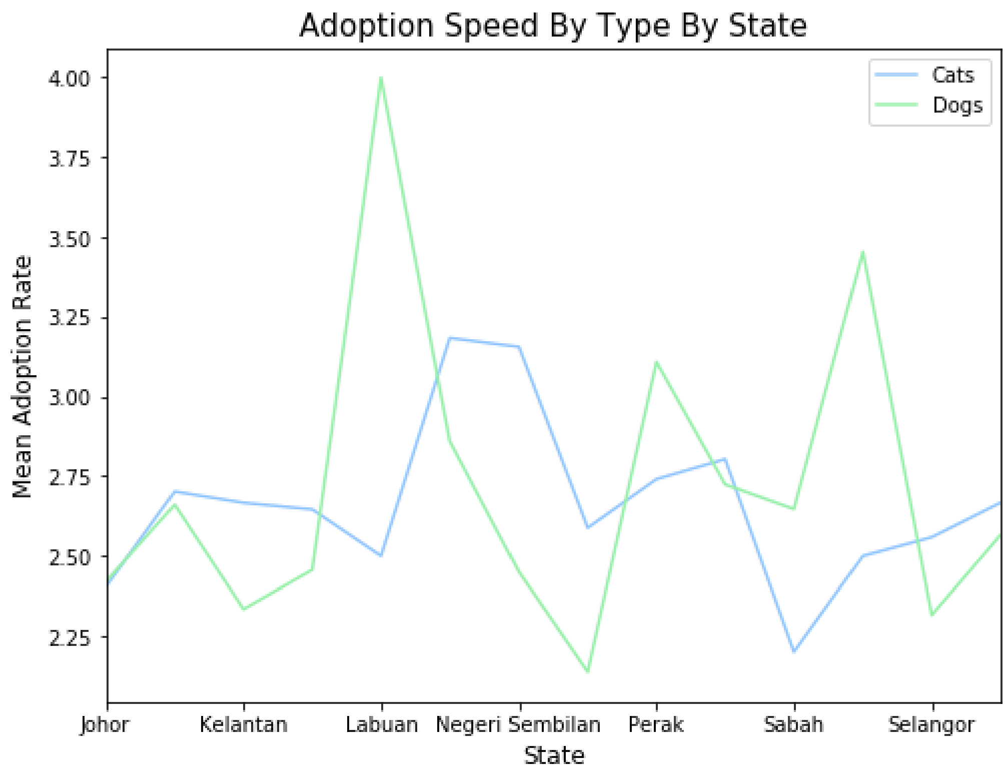


Within all the states, Pahang has the quickiest adoption speed (mean).

We also assessed adoption speed by dog vs. cat within states:

In [33]:
```
df.loc[df['type']==1,:].groupby('state_name')['adoptionspeed'].mean().pl
ot(figsize=(8,6))
df.loc[df['type']==2,:].groupby('state_name')['adoptionspeed'].mean().pl
ot(figsize=(8,6))
plt.title('Adoption Speed By Type By State',fontsize=15)
plt.xlabel('State',fontsize=12)
plt.ylabel('Mean Adoption Rate',fontsize=12)
plt.legend(['Cats','Dogs'])
```

Out[33]:
```
<matplotlib.legend.Legend at 0x1a1eb72dd8>
```



Some states have very different preferences for dogs vs cats. For example, Labuan and Sabah have high preference for cats.

We then analyzed the adpotion speed by age for cats and dogs.

In [34]:
```
df['age_yr']=df['age']/12
```

In [35]:
```
df.groupby(['age_yr','type']).agg({'adoptionspeed':'mean'})
```

Out[35]:

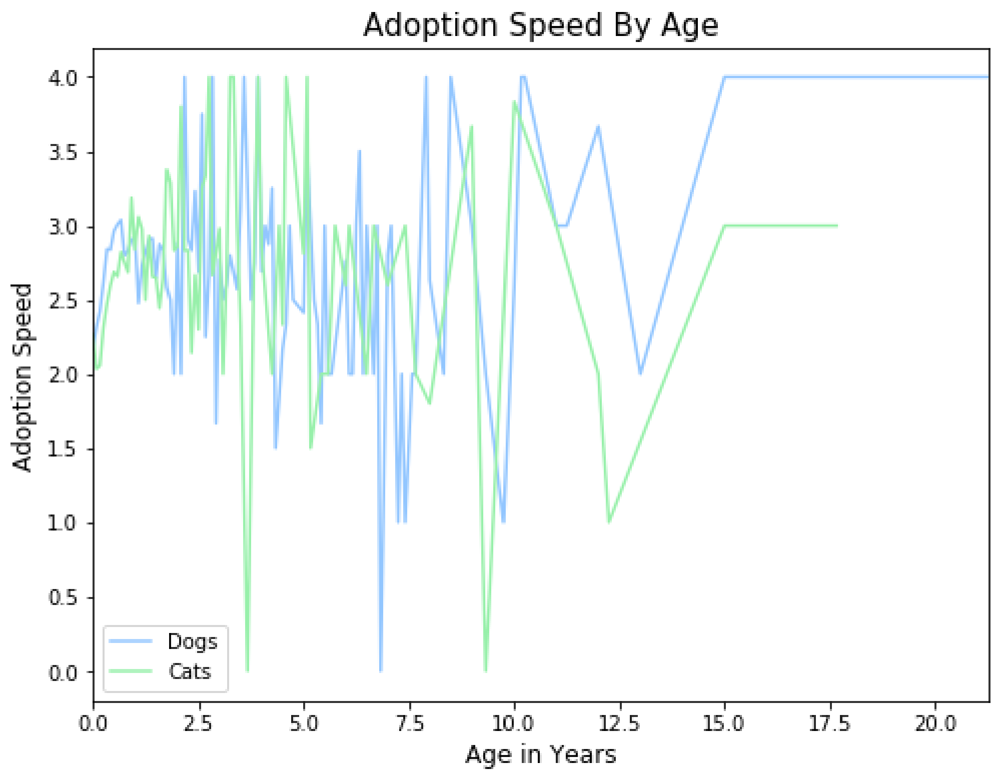
		adoptionspeed
age_yr	**type**	
0.000000	**1.0**	2.166667
	2.0	2.265957
0.083333	**1.0**	2.314685
	2.0	2.034517
0.166667	**1.0**	2.423387
	2.0	2.056106
0.250000	**1.0**	2.618337
	2.0	2.312865
0.333333	**1.0**	2.842105
	2.0	2.471609
0.416667	**1.0**	2.837500
	2.0	2.597183
0.500000	**1.0**	2.964126
	2.0	2.688623
0.583333	**1.0**	3.008403
	2.0	2.658385
0.666667	**1.0**	3.038217
	2.0	2.822368
0.750000	**1.0**	2.797980
	2.0	2.752941
0.833333	**1.0**	2.828947
	2.0	2.686047
0.916667	**1.0**	2.902439
	2.0	3.188679
1.000000	**1.0**	2.927386
	2.0	2.839175
1.083333	**1.0**	2.478261
	2.0	3.058824
1.166667	**1.0**	2.733333
	2.0	2.978261
...	**...**	...
7.666667	**1.0**	2.000000
	2.0	2.000000

		adoptionspeed
age_yr	**type**	
7.916667	**1.0**	4.000000
8.000000	**1.0**	2.634146
	2.0	1.800000
8.333333	**1.0**	2.000000
8.500000	**1.0**	4.000000
9.000000	**1.0**	3.000000
	2.0	3.666667
9.333333	**1.0**	2.000000
	2.0	0.000000
9.750000	**1.0**	1.000000
10.000000	**1.0**	2.592593
	2.0	3.833333
10.166667	**1.0**	4.000000
10.250000	**1.0**	4.000000
11.000000	**1.0**	3.000000
	2.0	3.000000
11.250000	**1.0**	3.000000
12.000000	**1.0**	3.666667
	2.0	2.000000
12.250000	**2.0**	1.000000
13.000000	**1.0**	2.000000
14.000000	**1.0**	3.000000
15.000000	**1.0**	4.000000
	2.0	3.000000
17.666667	**1.0**	4.000000
	2.0	3.000000
19.833333	**1.0**	4.000000
21.250000	**1.0**	4.000000

174 rows × 1 columns

In [36]:
```
df.loc[df['type']==1,:].groupby('age_yr')['adoptionspeed'].mean().plot(f
igsize=(8,6))
df.loc[df['type']==2,:].groupby('age_yr')['adoptionspeed'].mean().plot(f
igsize=(8,6))
plt.title('Adoption Speed By Age',fontsize=15)
plt.legend(['Dogs','Cats'])
plt.xlabel('Age in Years',fontsize=12)
plt.ylabel('Adoption Speed',fontsize=12)
```

Out[36]:
```
Text(0, 0.5, 'Adoption Speed')
```



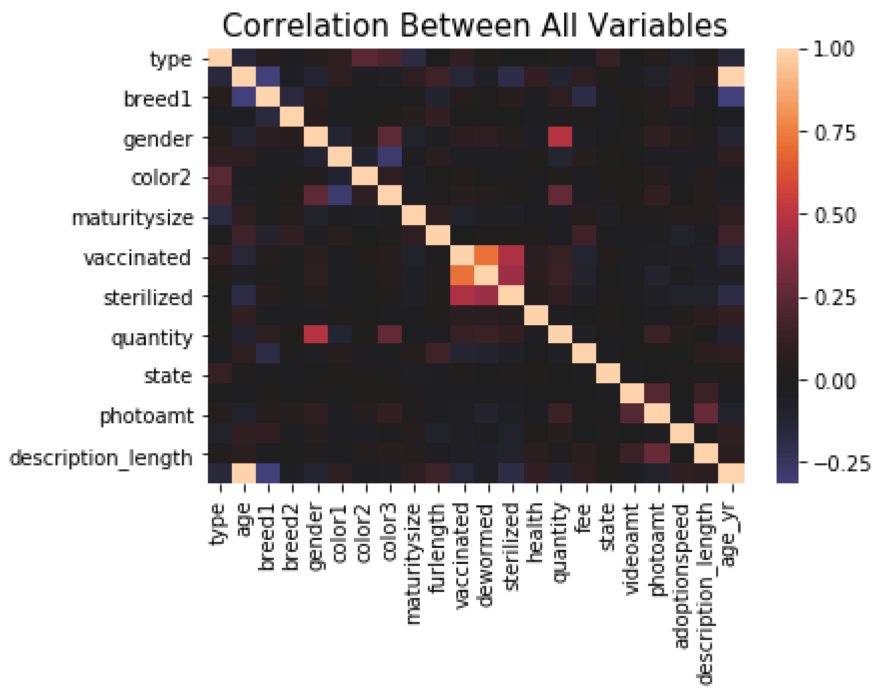
We can see that cats are more likely to be adopted around years 1 and 3, and are adopted quicker than dogs after age 4. Unfortunately dogs have a high chance of not being adopted after year 5.

REGRESSION ANALYSIS

Before beginning regression analysis, we assessed correlations between variables.

In [37]:
```
sns.heatmap(df.corr(),center=0,annot=False)
plt.title('Correlation Between All Variables',fontsize=15)
```

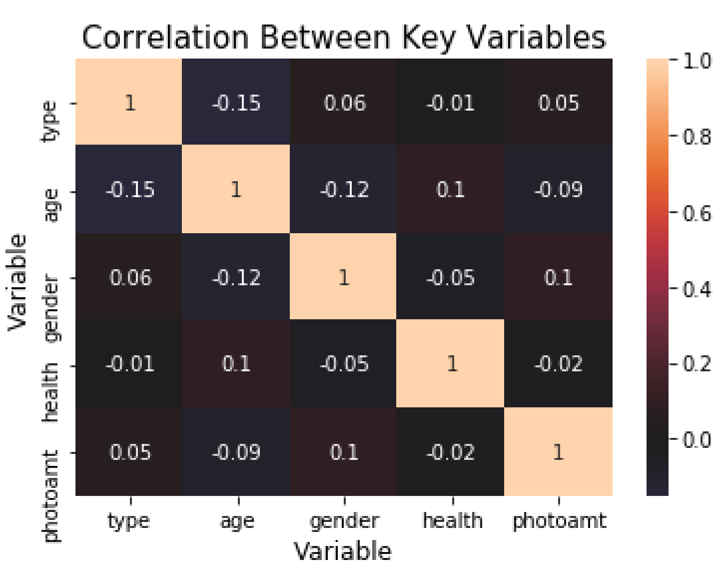
Out[37]:
```
Text(0.5, 1.0, 'Correlation Between All Variables')
```



This graphic is clearly too busy, so we show only certain columns.

In [38]:
```
sns.heatmap(round(df[['type','age','gender','health','photoamt']].corr
(),2),center=0,annot=True)
plt.title('Correlation Between Key Variables',fontsize=15)
plt.xlabel('Variable',fontsize=12)
plt.ylabel('Variable',fontsize=12)
```

Out[38]:
```
Text(33.0, 0.5, 'Variable')
```



We ran regressions to see which features contribute to adoption.

In [39]:
```
reg_type = smf.ols('adoptionspeed ~ type',data=df).fit()
print(reg_type.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:          adoptionspeed   R-squared:
0.008
Model:                            OLS   Adj. R-squared:
0.008
Method:                 Least Squares   F-statistic:
126.4
Date:                Fri, 05 Jul 2019   Prob (F-statistic):
3.23e-29
Time:                        21:01:08   Log-Likelihood:
-23634.
No. Observations:               14977   AIC:                              4.
727e+04
Df Residuals:                   14975   BIC:                              4.
729e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=======
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-------
Intercept      2.8315      0.030     95.591      0.000       2.773
2.890
type          -0.2163      0.019    -11.244      0.000      -0.254
-0.179
==============================================================================
=======
Omnibus:                     7190.984   Durbin-Watson:
2.004
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
837.088
Skew:                          -0.143   Prob(JB):                         1.
69e-182
Kurtosis:                       1.877   Cond. No.
6.62
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

The regression result shows this model using 'type' as the indepedent variable explains only 0.8% variability of the dependent variable 'adoptionspeed' (R-squared = 0.008).

The result is statistically siginificantly at either p = 0.05 or p = 0.025.

The coefficient is -0.2156, which means the adoption speed and pet type are slightly negatively related: people tend to adopt cats faster than dogs.

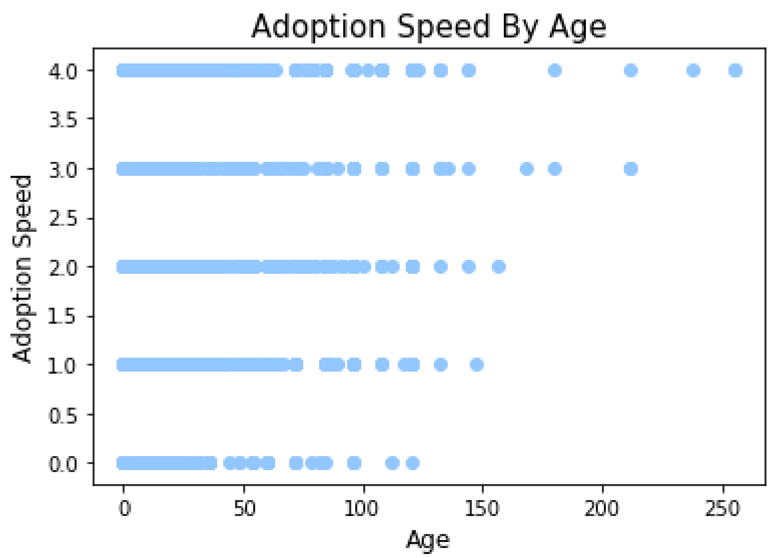
In [37]:
```
reg_age = smf.ols('adoptionspeed ~ age',data=df).fit()
print(reg_age.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:          adoptionspeed   R-squared:
0.010
Model:                            OLS   Adj. R-squared:
0.010
Method:                 Least Squares   F-statistic:
154.3
Date:                Fri, 05 Jul 2019   Prob (F-statistic):
2.99e-35
Time:                        14:10:57   Log-Likelihood:
-23621.
No. Observations:               14977   AIC:                              4.
725e+04
Df Residuals:                   14975   BIC:                              4.
726e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=======
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-------
Intercept      2.4479      0.011    221.620      0.000       2.426
2.470
age            0.0066      0.001     12.421      0.000       0.006
0.008
==============================================================================
=======
Omnibus:                     6288.843   Durbin-Watson:
2.006
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
827.259
Skew:                          -0.167   Prob(JB):                         2.
31e-180
Kurtosis:                       1.898   Cond. No.
24.2
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

In [38]:
```
plt.scatter(df['age'], df['adoptionspeed'])
plt.title('Adoption Speed By Age',fontsize=15)
plt.xlabel('Age',fontsize=12)
plt.ylabel('Adoption Speed',fontsize=12)
```

Out[38]:
```
Text(0, 0.5, 'Adoption Speed')
```



There are different ages (in months) of pets in this dataset. The regression result shows this model using 'age' as the indepedent viarable explains only 1% variability of the dependent variable 'adoptionspeed' (R-squared = 0.010).

The result is statistically siginificantly at p = 0.05 or p = 0.025.

The coefficient is 0.0065, which means the adoption speed and pet age are positively related: In general, older pets tend to take a bit longer time to adopt.

The plot chart also tells the same story.

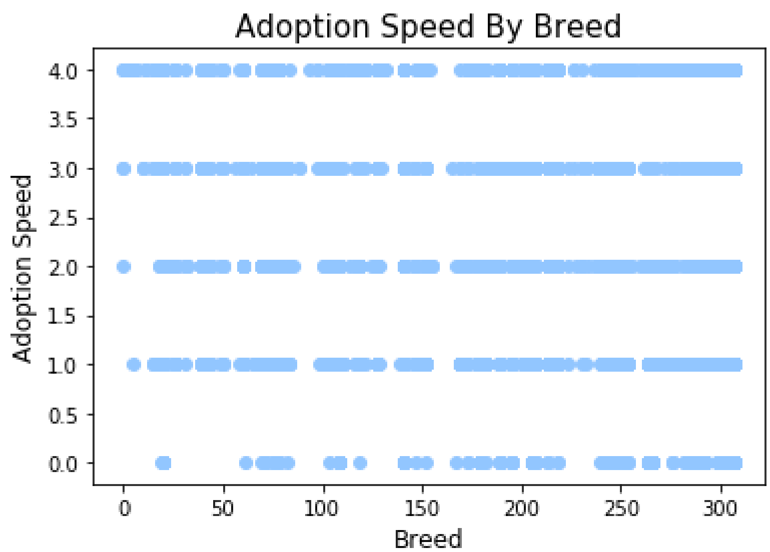
In [39]:
```
reg_breed1 = smf.ols('adoptionspeed ~ breed1',data=df).fit()
print(reg_breed1.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:          adoptionspeed   R-squared:
0.012
Model:                            OLS   Adj. R-squared:
0.012
Method:                 Least Squares   F-statistic:
175.8
Date:                Fri, 05 Jul 2019   Prob (F-statistic):
6.61e-40
Time:                        14:10:57   Log-Likelihood:
-23610.
No. Observations:               14977   AIC:                             4.
722e+04
Df Residuals:                   14975   BIC:                             4.
724e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=======
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-------
Intercept      1.9562      0.043     45.165      0.000       1.871
2.041
breed1         0.0021      0.000     13.260      0.000       0.002
0.002
==============================================================================
=======
Omnibus:                     7312.752   Durbin-Watson:
2.007
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
849.782
Skew:                          -0.155   Prob(JB):                        2.
97e-185
Kurtosis:                       1.875   Cond. No.
1.23e+03
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 1.23e+03. This might indicate that t
here are
strong multicollinearity or other numerical problems.
```

In [40]:
```
plt.scatter(df['breed1'], df['adoptionspeed'])
plt.title('Adoption Speed By Breed',fontsize=15)
plt.xlabel('Breed',fontsize=12)
plt.ylabel('Adoption Speed',fontsize=12)
```

Out[40]: Text(0, 0.5, 'Adoption Speed')



There are different primary breeds of pets in this dataset. The regression result shows this model using 'breed1' as the indepedent viarable explains only 1.2% variability of the dependent variable 'adoptionspeed' (R-squared = 0.012).

The result is statistically siginificantly at p = 0.05 or p = 0.025.

The coefficient is 0.0021, which means the adoption speed and pet age are slightly positively related. Higher code primary breeds tend to take longer time to adopt.

The plot chart also tells the same story.

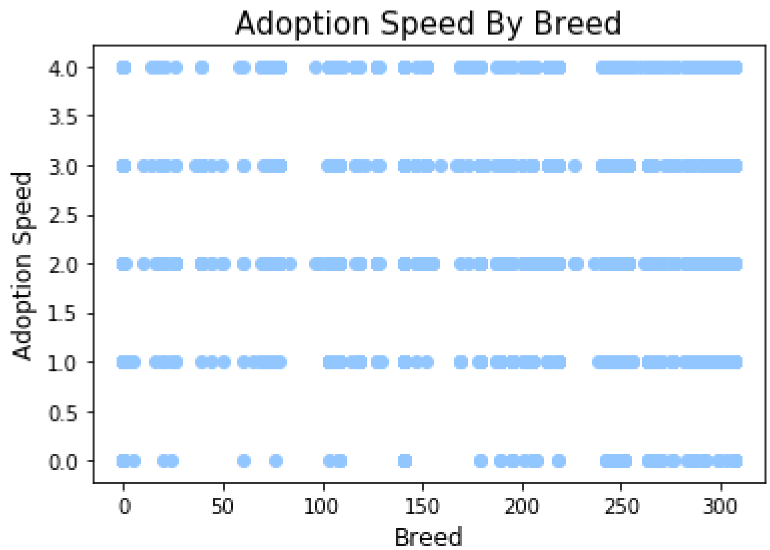
In [41]:
```
reg_breed2 = smf.ols('adoptionspeed ~ breed2',data=df).fit()
print(reg_breed2.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:          adoptionspeed   R-squared:
0.000
Model:                            OLS   Adj. R-squared:
0.000
Method:                 Least Squares   F-statistic:
5.360
Date:                Fri, 05 Jul 2019   Prob (F-statistic):
0.0206
Time:                        14:10:58   Log-Likelihood:
-23695.
No. Observations:               14977   AIC:                              4.
739e+04
Df Residuals:                   14975   BIC:                              4.
741e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=======
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-------
Intercept      2.5297      0.011    225.338      0.000       2.508
2.552
breed2        -0.0002   7.82e-05     -2.315      0.021      -0.000     -
2.78e-05
==============================================================================
=======
Omnibus:                     8219.427   Durbin-Watson:
2.007
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
873.141
Skew:                          -0.155   Prob(JB):                         2.
51e-190
Kurtosis:                       1.858   Cond. No.
168.
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

In [42]:
```
plt.scatter(df['breed2'], df['adoptionspeed'])
plt.title('Adoption Speed By Breed',fontsize=15)
plt.xlabel('Breed',fontsize=12)
plt.ylabel('Adoption Speed',fontsize=12)
```

Out[42]: Text(0, 0.5, 'Adoption Speed')



The regression result shows this model using 'breed2' as the indepedent variable explains 0 of the dependent variable 'adoptionspeed' (R-squared = 0.000).

In [43]:
```
reg_gender = smf.ols('adoptionspeed ~ gender',data=df).fit()
print(reg_gender.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:          adoptionspeed   R-squared:
0.003
Model:                            OLS   Adj. R-squared:
0.003
Method:                 Least Squares   F-statistic:
49.84
Date:                Fri, 05 Jul 2019   Prob (F-statistic):
1.74e-12
Time:                        14:10:58   Log-Likelihood:
-23672.
No. Observations:               14977   AIC:                             4.
735e+04
Df Residuals:                   14975   BIC:                             4.
736e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=======
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-------
Intercept      2.3396      0.027     87.243      0.000       2.287
2.392
gender         0.0995      0.014      7.060      0.000       0.072
0.127
==============================================================================
=======
Omnibus:                     7518.701   Durbin-Watson:
2.004
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
855.439
Skew:                          -0.155   Prob(JB):                        1.
75e-186
Kurtosis:                       1.871   Cond. No.
6.63
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

The regression result shows this model using 'gender' as the indepedent variable only explains 0.3% of the dependent variable 'adoptionspeed' (R-squared = 0.003).

In [44]:
```
reg_maturitysize = smf.ols('adoptionspeed ~ maturitysize',data=df).fit()
print(reg_maturitysize.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:           adoptionspeed   R-squared:
0.002
Model:                             OLS   Adj. R-squared:
0.002
Method:                  Least Squares   F-statistic:
32.34
Date:                 Fri, 05 Jul 2019   Prob (F-statistic):
1.32e-08
Time:                         14:10:58   Log-Likelihood:
-23681.
No. Observations:                14977   AIC:                          4.
737e+04
Df Residuals:                    14975   BIC:                          4.
738e+04
Df Model:                            1
Covariance Type:             nonrobust
==============================================================================
=========
                   coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
---------
Intercept        2.3305      0.034     68.436      0.000       2.264
2.397
maturitysize     0.0998      0.018      5.686      0.000       0.065
0.134
==============================================================================
=======
Omnibus:                      7880.082   Durbin-Watson:
2.007
Prob(Omnibus):                   0.000   Jarque-Bera (JB):
862.818
Skew:                           -0.152   Prob(JB):                     4.
38e-188
Kurtosis:                        1.864   Cond. No.
8.59
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

The regression result shows this model using 'maturitysize' as the indepedent variable only explains 0.2% of the dependent variable 'adoptionspeed' (R-squared = 0.002).

In [45]:
```
reg_vaccinated = smf.ols('adoptionspeed ~ vaccinated',data=df).fit()
print(reg_vaccinated.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:          adoptionspeed   R-squared:
0.004
Model:                            OLS   Adj. R-squared:
0.003
Method:                 Least Squares   F-statistic:
53.56
Date:                Fri, 05 Jul 2019   Prob (F-statistic):
2.64e-13
Time:                        14:10:58   Log-Likelihood:
-23671.
No. Observations:               14977   AIC:                             4.
735e+04
Df Residuals:                   14975   BIC:                             4.
736e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=======
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-------
Intercept      2.6986      0.027    101.095      0.000       2.646
2.751
vaccinated    -0.1053      0.014     -7.318      0.000      -0.133
-0.077
==============================================================================
=======
Omnibus:                     7453.500   Durbin-Watson:
2.008
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
850.354
Skew:                          -0.151   Prob(JB):                        2.
23e-185
Kurtosis:                       1.872   Cond. No.
6.50
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

Vaccinated (1 = Yes, 2 = No, 3 = Not Sure)

The regression result shows this model using 'vaccinated' as the indepedent variable only explains 0.3% of the dependent variable 'adoptionspeed' (R-squared = 0.003).

In [46]:
```python
reg_health = smf.ols('adoptionspeed ~ health',data=df).fit()
print(reg_health.summary())
```

```
                            OLS Regression Results                            
==============================================================================
=======
Dep. Variable:           adoptionspeed   R-squared:                       
0.001
Model:                             OLS   Adj. R-squared:                  
0.001
Method:                  Least Squares   F-statistic:                     
12.27
Date:                 Fri, 05 Jul 2019   Prob (F-statistic):             
0.000462
Time:                         14:10:58   Log-Likelihood:                  
-23691.
No. Observations:                14977   AIC:                          4.
739e+04
Df Residuals:                    14975   BIC:                          4.
740e+04
Df Model:                            1                                         
Covariance Type:             nonrobust                                         
==============================================================================
=======
                 coef    std err          t      P>|t|      [0.025      
0.975]
------------------------------------------------------------------------------
-------
Intercept      2.3411      0.051     45.968      0.000       2.241       
2.441
health         0.1690      0.048      3.503      0.000       0.074       
0.264
==============================================================================
=======
Omnibus:                      7847.208   Durbin-Watson:                   
2.007
Prob(Omnibus):                   0.000   Jarque-Bera (JB):              
864.925
Skew:                           -0.156   Prob(JB):                      1.
53e-188
Kurtosis:                        1.865   Cond. No.                        
10.5
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is 
correctly specified.
```

The regression result shows this model using 'health' as the indepedent variable only explains 0.1% of the dependent variable 'adoptionspeed' (R-squared = 0.001).

In [47]:
```
reg_videoamt = smf.ols('adoptionspeed ~ videoamt',data=df).fit()
print(reg_videoamt.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:          adoptionspeed   R-squared:
0.000
Model:                            OLS   Adj. R-squared:
-0.000
Method:                 Least Squares   F-statistic:
0.004621
Date:                Fri, 05 Jul 2019   Prob (F-statistic):
0.946
Time:                        14:10:58   Log-Likelihood:
-23697.
No. Observations:               14977   AIC:                             4.
740e+04
Df Residuals:                   14975   BIC:                             4.
741e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=======
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-------
Intercept      2.5164      0.010    258.093      0.000       2.497
2.536
videoamt      -0.0019      0.028     -0.068      0.946      -0.056
0.053
==============================================================================
=======
Omnibus:                     8110.175   Durbin-Watson:
2.006
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
870.551
Skew:                          -0.155   Prob(JB):                        9.
17e-190
Kurtosis:                       1.860   Cond. No.
2.90
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

The regression result shows this model using 'videoamt' as the indepedent viarable explains 0.000 of the model and is not statistically signigicant.

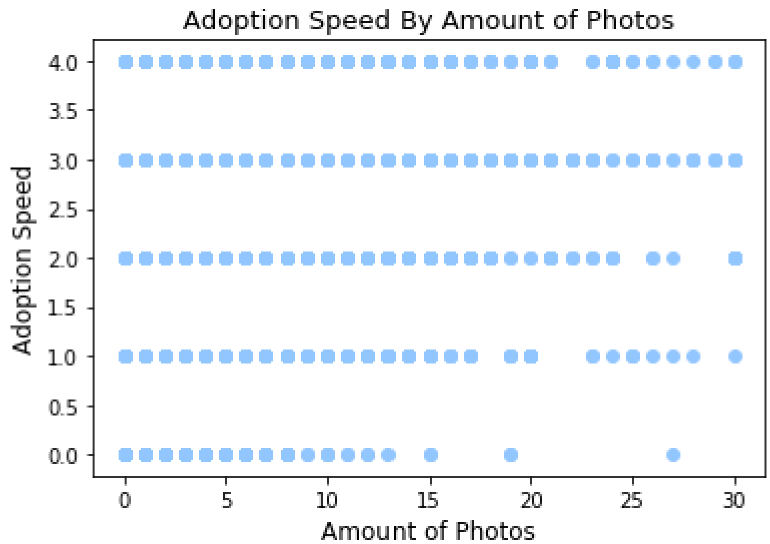
In [48]:
```
reg_photoamt = smf.ols('adoptionspeed ~ photoamt',data=df).fit()
print(reg_photoamt.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:          adoptionspeed   R-squared:
0.001
Model:                            OLS   Adj. R-squared:
0.000
Method:                 Least Squares   F-statistic:
7.951
Date:                Fri, 05 Jul 2019   Prob (F-statistic):
0.00481
Time:                        14:10:58   Log-Likelihood:
-23693.
No. Observations:               14977   AIC:                              4.
739e+04
Df Residuals:                   14975   BIC:                              4.
741e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=======
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-------
Intercept      2.5466      0.014    176.752      0.000       2.518
2.575
photoamt      -0.0078      0.003     -2.820      0.005      -0.013
-0.002
==============================================================================
=======
Omnibus:                     7985.070   Durbin-Watson:
2.006
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
872.646
Skew:                          -0.162   Prob(JB):                         3.
22e-190
Kurtosis:                       1.863   Cond. No.
7.99
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

In [49]:
```
plt.scatter(df['photoamt'], df['adoptionspeed'])
plt.title('Adoption Speed By Amount of Photos',fontsize=13)
plt.xlabel('Amount of Photos',fontsize=12)
plt.ylabel('Adoption Speed',fontsize=12)
```

Out[49]: Text(0, 0.5, 'Adoption Speed')



The regression result shows this model using 'photoamt' as the indepedent variable only explains 0.1% of the dependent variable 'adoptionspeed' (R-squared = 0.001).

In [50]:
```
reg_fee = smf.ols('adoptionspeed ~ fee',data=df).fit()
print(reg_fee.summary())
```

```
                            OLS Regression Results                            
==============================================================================
=======
Dep. Variable:          adoptionspeed   R-squared:                       
0.000
Model:                            OLS   Adj. R-squared:                  
-0.000
Method:                 Least Squares   F-statistic:                     
0.2083
Date:                Fri, 05 Jul 2019   Prob (F-statistic):              
0.648
Time:                        14:10:58   Log-Likelihood:                  
-23697.
No. Observations:               14977   AIC:                          4.
740e+04
Df Residuals:                   14975   BIC:                          4.
741e+04
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
==============================================================================
=======
                 coef    std err          t      P>|t|      [0.025      
0.975]
------------------------------------------------------------------------------
-------
Intercept      2.5175      0.010    252.540      0.000       2.498       
2.537
fee        -5.598e-05      0.000     -0.456      0.648      -0.000       
0.000
==============================================================================
=======
Omnibus:                     8106.690   Durbin-Watson:                   
2.006
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              
870.415
Skew:                          -0.155   Prob(JB):                     9.
81e-190
Kurtosis:                       1.860   Cond. No.                        
84.2
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is 
correctly specified.
```

The regression result shows this model using 'fee' as the indepedent variable explains 0.000 of the model and is not statistically significant.

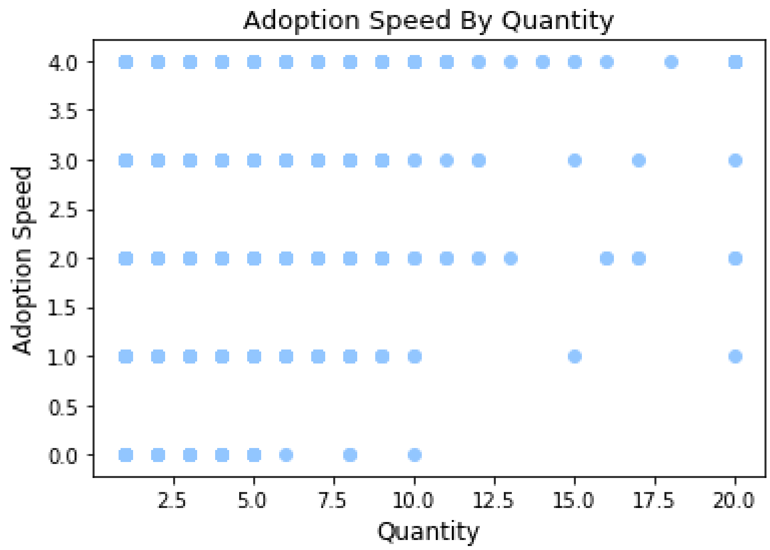
In [51]:
```
reg_quantity = smf.ols('adoptionspeed ~ quantity',data=df).fit()
print(reg_quantity.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:          adoptionspeed   R-squared:
0.004
Model:                            OLS   Adj. R-squared:
0.004
Method:                 Least Squares   F-statistic:
59.24
Date:                Fri, 05 Jul 2019   Prob (F-statistic):
1.48e-14
Time:                        14:10:58   Log-Likelihood:
-23668.
No. Observations:               14977   AIC:                             4.
734e+04
Df Residuals:                   14975   BIC:                             4.
735e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=======
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-------
Intercept      2.4372      0.014    173.288      0.000       2.410
2.465
quantity       0.0502      0.007      7.697      0.000       0.037
0.063
==============================================================================
=======
Omnibus:                     7734.273   Durbin-Watson:
2.006
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
860.928
Skew:                          -0.155   Prob(JB):                        1.
13e-187
Kurtosis:                       1.867   Cond. No.
3.56
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

In [52]:
```
plt.scatter(df['quantity'], df['adoptionspeed'])
plt.title('Adoption Speed By Quantity',fontsize=13)
plt.xlabel('Quantity',fontsize=12)
plt.ylabel('Adoption Speed',fontsize=12)
```

Out[52]:
```
Text(0, 0.5, 'Adoption Speed')
```



The regression result shows this model using 'quantity' as the indepedent variable only explains 0.4% of the dependent variable 'adoptionspeed' (R-squared = 0.004).

In [53]:
```
df.columns
```

Out[53]:
```
Index(['type', 'age', 'breed1', 'breed2', 'gender', 'color1', 'color2',
       'color3', 'maturitysize', 'furlength', 'vaccinated', 'dewormed',
       'sterilized', 'health', 'quantity', 'fee', 'state', 'rescuerid',
       'videoamt', 'description', 'petid', 'photoamt', 'adoptionspeed',
       'description_length', 'state_name', 'age_yr'],
      dtype='object')
```

In [54]:
```
reg_total = smf.ols('adoptionspeed ~ type + age + breed1 + gender + quan
tity + maturitysize + vaccinated + dewormed + sterilized + health + phot
oamt',data=df).fit()
print(reg_total.summary())
```

```
                            OLS Regression Results                            
==============================================================================
=======
Dep. Variable:          adoptionspeed   R-squared:                       
0.053
Model:                            OLS   Adj. R-squared:                  
0.053
Method:                 Least Squares   F-statistic:                     
76.63
Date:                Fri, 05 Jul 2019   Prob (F-statistic):          8.
32e-169
Time:                        14:10:59   Log-Likelihood:                 
-23287.
No. Observations:               14977   AIC:                             4.
660e+04
Df Residuals:                   14965   BIC:                             4.
669e+04
Df Model:                          11                                         
Covariance Type:            nonrobust                                         
==============================================================================
=========
                   coef    std err          t      P>|t|      [0.025      
0.975]
------------------------------------------------------------------------------
---------
Intercept        1.7848      0.093     19.293      0.000       1.603      
1.966
type            -0.1801      0.019     -9.278      0.000      -0.218      
-0.142
age              0.0080      0.001     14.026      0.000       0.007      
0.009
breed1           0.0030      0.000     17.938      0.000       0.003      
0.003
gender           0.0840      0.016      5.270      0.000       0.053      
0.115
quantity         0.0444      0.007      5.973      0.000       0.030      
0.059
maturitysize     0.0531      0.018      3.036      0.002       0.019      
0.087
vaccinated      -0.0965      0.021     -4.534      0.000      -0.138      
-0.055
dewormed         0.0803      0.020      4.033      0.000       0.041      
0.119
sterilized      -0.1511      0.019     -7.839      0.000      -0.189      
-0.113
health           0.1737      0.048      3.654      0.000       0.081      
0.267
photoamt        -0.0102      0.003     -3.709      0.000      -0.016      
-0.005
==============================================================================
=======
Omnibus:                     3545.756   Durbin-Watson:                   
2.006
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               
688.542
Skew:                          -0.156   Prob(JB):                        3.
05e-150
```

```
Kurtosis:                       1.997   Cond. No.
2.82e+03
=======================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 2.82e+03. This might indicate that t
here are
strong multicollinearity or other numerical problems.
```

This reg_total model takes multiple independent variables which were tested to be statistically significant in impacting the dependent variable 'adoptionspeed'.

This model explains 5.3% of the total variable in the independent variables, which is still a small amount.

Based on the coefficient of each independent variable, dogs tend to be more likely to be adopted than cats, younger pets tend to be more likely to be adopted than older pets, and higher code primary breeds tend to be more likely to be adopted than lower code primary breeds.

MACHINE LEARNING ALGORITHMS

We then took our analysis a step beyond regression by assessing machine learning algorithms for pet predictions.

In [40]:
```
y,X = patsy.dmatrices('adoptionspeed ~ type + age + breed1 + gender + qu
antity + maturitysize + vaccinated + dewormed + sterilized + health + ph
otoamt',
                      data=df)
```

In [41]:
```
reg().fit(X,y).score(X,y)
```

Out[41]: 0.05332327228800893

In [116]:
```
cross_val_score(reg(),X,np.ravel(y),cv=5).mean()
```

Out[116]: 0.05364888747525944

We found R squared for Multiple Linear Regression - 5.36%.

In [42]:
```
knn().fit(X,y).score(X,y)
```

Out[42]: 0.30472111310440975

We found R Squared for K-Nearest Neighbors. (We are aware that the below score can be based on an overfitted KNN model)

To alleviate the overfitting problem, we used 5-fold cross validation to get a more realistic score.

In [43]:
```
cross_val_score(knn(),X,np.ravel(y),cv=5).mean()
```

Out[43]:
```
0.015921865420894642
```

When using cross validation, the result from the knn model is worse than the one from the regression model.

Thus, we try to find the optimized K value.

In [59]:
```
for i in range(1,200):
    print(i,cross_val_score(knn(i),X,np.ravel(y),cv=5).mean())
```

```
1 -0.6214057063532189
2 -0.20923041773963363
3 -0.08801622920708754
4 -0.025523826993453725
5 0.015921865420894642
6 0.03790874165342599
7 0.05009829955079057
8 0.06331616130137274
9 0.07141700594656217
10 0.07941288455201763
11 0.08544553418683645
12 0.09001386228372277
13 0.09230711556213685
14 0.09418064868487748
15 0.0960684604776973
16 0.0991058101507494
17 0.1014389061963675
18 0.10245635555220758
19 0.10332018484620105
20 0.10679298851858354
21 0.10667170182699515
22 0.10740552052542558
23 0.1102315655193015
24 0.11109406020565493
25 0.11166573894573098
26 0.11194193916787909
27 0.11297866089875001
28 0.11435264662108904
29 0.11536265185724022
30 0.11694407518449568
31 0.11614719854392445
32 0.11637698930912128
33 0.11546926452028838
34 0.11672187579209703
35 0.11915051297117363
36 0.1191304148394599
37 0.11979554891729108
38 0.1206003342551107
39 0.12023544329844053
40 0.12157847453095905
41 0.12070114740106655
42 0.1221155409468027
43 0.12223459817614832
44 0.12175388423923664
45 0.12139179057585399
46 0.12214640291858367
47 0.12258126098814004
48 0.12263586432887905
49 0.12257155643246857
50 0.1231784029083427
51 0.12399626879703925
52 0.12416591418996117
53 0.12367956370240185
54 0.1244178172927736
55 0.12468509537464952
56 0.12422241222004489
57 0.12438557065580444
```

```
58 0.12445561894249964
59 0.12412903509631854
60 0.12418179403635998
61 0.12447628324645425
62 0.12454979265317052
63 0.12423441476567279
64 0.12467289587757757
65 0.12432114139853434
66 0.12463260983654886
67 0.12508817561613433
68 0.1251420899487351
69 0.1248307658338593
70 0.12488221748707584
71 0.12477401351327752
72 0.12360936184529489
73 0.12381830764239916
74 0.12427723152196941
75 0.12316568108690777
76 0.12354078594872833
77 0.12331474149444632
78 0.1233841989130243
79 0.12360010741161823
80 0.12285275472477897
81 0.12309910738423864
82 0.12321342949104071
83 0.12310557037977014
84 0.12321556238722364
85 0.12373747502522178
86 0.1233658261157077
87 0.12349186199866473
88 0.1236259936871544
89 0.12296174447565777
90 0.12376826077036795
91 0.12369498229535829
92 0.12319242069243237
93 0.12344200003322547
94 0.12351185530892013
95 0.12337730480921645
96 0.1229909270176139
97 0.12313607910130328
98 0.122817160458266
99 0.12295451222608464
100 0.12285189910516847
101 0.12271719349027628
102 0.12275029506134336
103 0.12267124063569332
104 0.12243543979859453
105 0.12221224635878877
106 0.12204759841984422
107 0.12193679288224064
108 0.12179924017593353
109 0.12115057711136187
110 0.1214789802703146
111 0.12163391732228987
112 0.12184829083130147
113 0.12128981406889909
114 0.12183827855830691
```

```
115 0.12166030509557421
116 0.12155824337817381
117 0.12158278645849299
118 0.12130917757094388
119 0.12135061662419563
120 0.12203127864595922
121 0.12152558597125315
122 0.12158696924632006
123 0.12158487083100904
124 0.12192038591307626
125 0.1219296188467824
126 0.12191071925275745
127 0.1221431019792015
128 0.12169833335175331
129 0.12170302727328675
130 0.12168475538195536
131 0.12161862353761721
132 0.12133298053754264
133 0.12134151881611806
134 0.12145941616549327
135 0.12160611196156228
136 0.1213167536508388
137 0.12161816114451822
138 0.12172410486928756
139 0.121314029678031 8
140 0.12152811973745108
141 0.12167681404071225
142 0.1213859657750371
143 0.12132629964873938
144 0.12112087358948564
145 0.12118102108524206
146 0.12098583158776795
147 0.12115494278381363
148 0.121114509955812
149 0.12135271795653942
150 0.12104875072068504
151 0.12112358627709896
152 0.12095284667827053
153 0.12086506687239676
154 0.12093859018754013
155 0.12091444226093298
156 0.12081531212178989
157 0.12078113434286655
158 0.12063786793813913
159 0.12060582281898453
160 0.12052831571597403
161 0.12042548600619066
162 0.12076479626077512
163 0.12074290539876528
164 0.12072823555304028
165 0.12091627474119485
166 0.12113197721936012
167 0.1209681195939272
168 0.12102610996410539
169 0.1209464644034826
170 0.12077372701421822
171 0.12054164719760149
```

```
172 0.12034597851767952
173 0.12034962986940408
174 0.12017605813390564
175 0.1204588066876223
176 0.12009768832796619
177 0.12026702888662286
178 0.12022208663668825
179 0.11991974731428516
180 0.11994419646910826
181 0.1200666770072166
182 0.11975057372788189
183 0.11990976170619976
184 0.11984247701383469
185 0.11974842243587572
186 0.11960581907981467
187 0.11965929455600119
188 0.11938567870772403
189 0.11954850253037955
190 0.11936224742773549
191 0.11933259297156681
192 0.11931382096382906
193 0.11962450050722465
194 0.11936213293468652
195 0.11925476877775795
196 0.11929581406624612
197 0.11907047670291164
198 0.11920827989760625
199 0.11937681937196103
```

We found that n=68 neighbors is the optimal amount for our model. When k=68, R Squared is 0.1251420899487351, higher than the score from the Regression model.

Next, we found R Squared using a Random Forest model.

In [60]:
```
cross_val_score(rf(n_estimators=100),X,np.ravel(y),cv=5).mean()
```

Out[60]:
```
0.031384833950204596
```

In [44]:
```
cross_val_score(rf(n_estimators=200),X,np.ravel(y),cv=5).mean()
```

Out[44]:
```
0.03273916233891825
```

We tuned the parameters for our Random Forest analysis.

In [46]:
```
for i in range(6,15):
    print(i, cross_val_score(rf(n_estimators=200, max_depth = i),X,np.ravel(y),cv=5).mean())
```

```
6 0.14142104290300656
7 0.14369979802185048
8 0.14541075683458107
9 0.14558445915743295
10 0.1454277127702301
11 0.1421095796622080
12 0.13825738136387092
13 0.13160497482829078
14 0.12248142739292663
```

Now we have found the optimal max_depth (max_depth = 9), we will tune for max_features.

In [47]:
```
for i in range(4,10):
    print(i, cross_val_score(rf(n_estimators=200, max_depth = 9, max_features = i),X,np.ravel(y),cv=5).mean())
```

```
4 0.15319127331770724
5 0.15369130318740476
6 0.15250857881472452
7 0.15235347089213444
8 0.1505303384243229
9 0.15005347828748147
```

Now we have found the optimal hyperparameters and the resulted R Sqaured for the Random Forest model is 0.15369130318740476, highest among all models we have tried.

We also decided to test the model via classification, where we segmented the "y" into two categories - (1) a pet is adopted within 100 days (adoption speeds between 0 and 3), (2) a pet is not adopted (adoption speed 4).

We denoted "1" for a pet adopted within 100 days and '0' for a pet not adopted.

In [51]:
```
df['adoption_indicator'] = np.where(df['adoptionspeed']==4,0,1)
```

In [69]:
```
y,X = patsy.dmatrices('adoption_indicator ~ type + age + breed1 + gender + quantity + maturitysize + vaccinated + dewormed + sterilized + health + photoamt',
                      data=df)
```

In [70]:
```
y = np.ravel(y)
```

In [71]:
```
yhat = logistic().fit(X,y).predict(X)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
```

In [72]:
```
pd.Series(yhat).value_counts()
```

Out[72]:
```
1.0    14456
0.0      521
dtype: int64
```

In [73]:
```
df['adoption_indicator'].value_counts()
```

Out[73]:
```
1    10784
0     4193
Name: adoption_indicator, dtype: int64
```

In [74]:
```
df['yhat'] = logistic().fit(X,y).predict(X)
df.pivot_table(index='adoption_indicator',columns='yhat',values='rescuer
id',aggfunc='count')/len(df)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
```

Out[74]:

yhat	0.0	1.0
adoption_indicator		
0	0.018028	0.261935
1	0.016759	0.703278

In [84]:
```
cross_val_score(logistic(),X,y,cv=5).mean()
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
```

Out[84]:
```
0.7215070835067304
```

The "confusion matrix" above shows that the model correctly predicts adoption only 72% of the time, using a logistic regression. Cross validation using the regression classifier model also returns a similar result.

We then analyzed classification under k-nearest neighbors.

In [75]:
```
pd.Series(knnc().fit(X,y).predict(X)).value_counts()
```

Out[75]:
```
1.0    12032
0.0     2945
dtype: int64
```

In [76]:
```
knnc().fit(X,y).score(X,y)
```

Out[76]:
```
array(0.79915871)
```

The above result shows that a knn model returns a model with almost 80% accuracy. (We are aware that this model may have overfitting problems)

In [77]:
```
df['yhat_knn']=knnc().fit(X,y).predict(X)
df
```

Out[77]:

	type	age	breed1	breed2	gender	color1	color2	color3	maturitysize	furlength	...	
0	2.0	3	299	0	1.0	1	7	0	1	1	...	o
1	2.0	1	265	0	1.0	1	2	0	2	2	...	al
2	1.0	1	307	0	1.0	2	7	0	2	2	...	- d
3	1.0	4	307	0	2.0	1	2	0	2	1	...	c
4	1.0	1	307	0	1.0	1	0	0	2	1	...	T
5	2.0	3	266	0	2.0	5	6	0	2	1	...	ki tc
6	2.0	12	264	264	1.0	1	0	0	2	3	...	th or
7	1.0	0	307	0	2.0	1	2	7	2	1	...	Si bi
8	2.0	2	265	0	2.0	6	0	0	2	2	...	
9	2.0	12	265	0	2.0	1	7	0	2	2	...	V g
10	1.0	2	307	0	1.0	1	2	7	2	1	...	a
11	2.0	3	264	0	2.0	1	2	5	3	3	...	
12	1.0	2	307	0	1.0	2	5	6	2	3	...	F
13	2.0	2	265	0	3.0	1	6	7	1	2	...	- p

	type	age	breed1	breed2	gender	color1	color2	color3	maturitysize	furlength	...	
14	1.0	3	307	0	2.0	2	5	7	2	2	...	Lo
15	1.0	78	218	205	1.0	1	7	0	2	2	...	of
16	2.0	6	266	0	2.0	2	0	0	1	1	...	to /
17	1.0	8	307	307	2.0	2	0	0	2	1	...	
18	1.0	2	307	0	2.0	1	0	0	2	1	...	c
19	2.0	1	266	0	3.0	1	2	7	1	1	...	E
20	1.0	12	307	0	2.0	2	7	0	2	2	...	
21	1.0	3	307	0	2.0	6	0	0	2	1	...	a
22	2.0	0	114	0	3.0	3	6	7	2	2	...	My
23	1.0	10	307	117	2.0	1	2	7	2	2	...	f
24	2.0	3	266	0	1.0	2	7	0	1	1	...	
25	2.0	36	285	251	1.0	3	0	0	3	2	...	Ga lar
26	2.0	2	285	265	1.0	3	0	0	2	2	...	
27	2.0	1	266	0	2.0	1	0	0	2	1	...	J l

	type	age	breed1	breed2	gender	color1	color2	color3	maturitysize	furlength	...	
28	1.0	14	189	0	1.0	1	2	0	3	1	...	a
29	2.0	1	266	0	2.0	2	7	0	1	1	...	Th a
...	...	...	...	...	...	...	...	...	...	...	...	
14947	1.0	48	83	0	1.0	5	0	0	1	2	...	
14948	2.0	1	303	0	2.0	1	2	7	1	2	...	t
14949	1.0	7	182	0	1.0	1	2	0	1	1	...	ac
14950	1.0	1	218	307	2.0	1	2	0	2	1	...	a
14951	2.0	6	276	0	1.0	1	0	0	2	2	...	r
14952	2.0	36	265	0	1.0	6	7	0	2	2	...	
14953	2.0	2	266	0	2.0	1	0	0	1	1	...	
14954	2.0	1	265	0	1.0	1	2	0	1	2	...	cu t
14955	2.0	24	265	0	2.0	2	4	0	1	2	...	
14956	2.0	2	266	0	3.0	3	7	0	2	1	...	PL Ca k
14957	2.0	10	266	0	1.0	1	7	0	2	1	...	fe

	type	age	breed1	breed2	gender	color1	color2	color3	maturitysize	furlength	...
14958	1.0	2	307	0	2.0	2	5	0	2	1	...
14959	2.0	5	265	0	1.0	3	7	0	3	2	...
14960	2.0	2	266	0	1.0	6	0	0	2	1	...
14961	1.0	2	307	307	2.0	2	7	0	2	2	...
14962	1.0	2	307	0	1.0	1	2	7	2	1	...
14963	1.0	24	307	0	2.0	2	7	0	2	2	...
14964	2.0	84	264	264	3.0	1	7	0	2	2	...
14965	2.0	3	254	0	2.0	1	2	7	1	2	...
14966	1.0	4	307	0	2.0	2	0	0	2	1	...
14967	1.0	6	307	0	1.0	1	7	0	2	1	...
14968	1.0	24	307	307	2.0	2	0	0	3	1	...
14969	1.0	8	307	0	2.0	2	7	0	2	2	...
14970	2.0	2	266	0	2.0	1	4	7	2	1	...
14971	1.0	60	307	0	2.0	2	5	0	2	2	...
14972	1.0	24	179	307	1.0	2	3	7	2	2	...

	type	age	breed1	breed2	gender	color1	color2	color3	maturitysize	furlength	...	
14973	1.0	6	195	0	2.0	1	7	0	1	3	...	
14974	2.0	2	266	0	3.0	1	0	0	2	2	...	I tl a
14975	2.0	60	265	264	3.0	1	4	7	2	2	...	ca y
14976	2.0	9	266	0	2.0	4	7	0	1	1	...	sl inc

14977 rows × 29 columns

In [78]:
```
df.pivot_table(index='adoption_indicator',columns='yhat_knn',values='res
cuerid',aggfunc='count')/len(df)
```

Out[78]:

yhat_knn	0.0	1.0
adoption_indicator		
0	0.137878	0.142085
1	0.058757	0.661281

The "confusion matrix" above shows that the model correctly predicts adoption 80% of the time. (We are aware that this model may have overfitting problems)

Now we wanted to tune for the optimal k value.

In [85]:
```
for i in range(50,70):
    print(i,cross_val_score(knnc(i),X,np.ravel(y),cv=5).mean())
```

```
50 0.7410027054055359
51 0.7424049536120936
52 0.7426053097909497
53 0.7428057552004107
54 0.7428057105627747
55 0.7420043080619441
56 0.742872421629149
57 0.742004151993998
58 0.7421380200713237
59 0.7430058438363436
60 0.7418039964313892
61 0.7415368845634116
62 0.7432729111113525
63 0.7432066236260692
64 0.7419376192101638
65 0.7433400235293348
66 0.742805844341681
67 0.7416708195227457
68 0.743072755593411
69 0.7419377753972229
```

After tuning the parameters, the model returns the best accuracy when n_neighbors = 65. The accuracy of the model is 0.7433400235293348.

We then analyzed classification under random forest.

In [113]:
```
yhat_rf = rfc().fit(X,y).predict(X)
df['yhat_rf']=rfc().fit(X,y).predict(X)
df.pivot_table(index='adoption_indicator',columns='yhat_rf',values='resc
uerid',aggfunc='count')/len(df)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:246:
FutureWarning: The default value of n_estimators will change from 10 in
version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:246:
FutureWarning: The default value of n_estimators will change from 10 in
version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[113]:

yhat_rf	0.0	1.0
adoption_indicator		
0	0.225212	0.054751
1	0.025239	0.694799

The "confusion matrix" above shows that the random forest classifier model correctly predicts adoption 95% of the time!

However, the high accuracy can be due to overfitting. We then want to use cross validation to find out the realistic accuracy.

In [86]:
```
cross_val_score(rfc(),X,y,cv=5).mean()
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:246:
FutureWarning: The default value of n_estimators will change from 10 in
version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:246:
FutureWarning: The default value of n_estimators will change from 10 in
version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:246:
FutureWarning: The default value of n_estimators will change from 10 in
version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:246:
FutureWarning: The default value of n_estimators will change from 10 in
version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:246:
FutureWarning: The default value of n_estimators will change from 10 in
version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[86]: 0.7163651799476595

The untuned random forest model returns an accuracy rate of 71.64%.

Now we want to tune for the hyperparameters.

In [87]:
```
cross_val_score(rfc(n_estimators=100),X,np.ravel(y),cv=5).mean()
```

Out[87]: 0.7268477158320794

In [88]:
```
cross_val_score(rfc(n_estimators=200),X,np.ravel(y),cv=5).mean()
```

Out[88]: 0.7289846777323545

In [90]:
```
for i in range(10,20):
    print(i, cross_val_score(rfc(n_estimators=200, max_depth = i),X,np.r
avel(y),cv=5).mean())
```

```
10 0.7534222482587587
11 0.7564931653116129
12 0.7555584743676049
13 0.7522867775250253
14 0.7522200886881341
15 0.747345720682356
16 0.7481468109728487
17 0.7444746694345932
18 0.7406683920527516
19 0.7375300951432157
```

In [92]:
```
for i in range(1,10):
    print(i, cross_val_score(rfc(n_estimators=200, max_depth = 11, max_f
eatures = i),X,np.ravel(y),cv=5).mean())
```

```
1 0.7454764280398343
2 0.7506849089012623
3 0.7547572056903481
4 0.7562932104248082
5 0.7552917861104967
6 0.7527543793507654
7 0.751753311780204
8 0.7531550917680342
9 0.7523538230312207
```

After tuning the parameters, the model returns the best accuracy when n_estimators=200, max_depth=11, and max_features = 4. The accuracy is 0.7564931653116129.

CONCLUSION

By tuning several machine learning models, we eventually landed on a model (random forest) that has 75.65% accuracy rate in predicting whether or not a pet will be adopted.