## horizontal line



BD Project

**Team 15**

|  |  |  |
| --- | --- | --- |
| **Name** | **Sec** | **B.N.** |
| Reem Emad | 1 | 33 |
| Osama Magdy | 1 | 14 |
| Ziad Atef | 1 | 35 |
| Yousef Gamal | 2 | 39 |

# 

# Overview

Millions of stray animals suffer on the streets or in shelters every day around the world. If homes can be found for them, many precious lives can be saved and more happy families created. Animal adoption rates are strongly correlated to the metadata associated with [PetFinder.](https://petfinder.my/) Using this [dataset](https://www.kaggle.com/competitions/petfinder-adoption-prediction/data), this problem aims to predict the speed at which a pet is adopted.

# Project Pipeline

The pipeline is split into two stages:

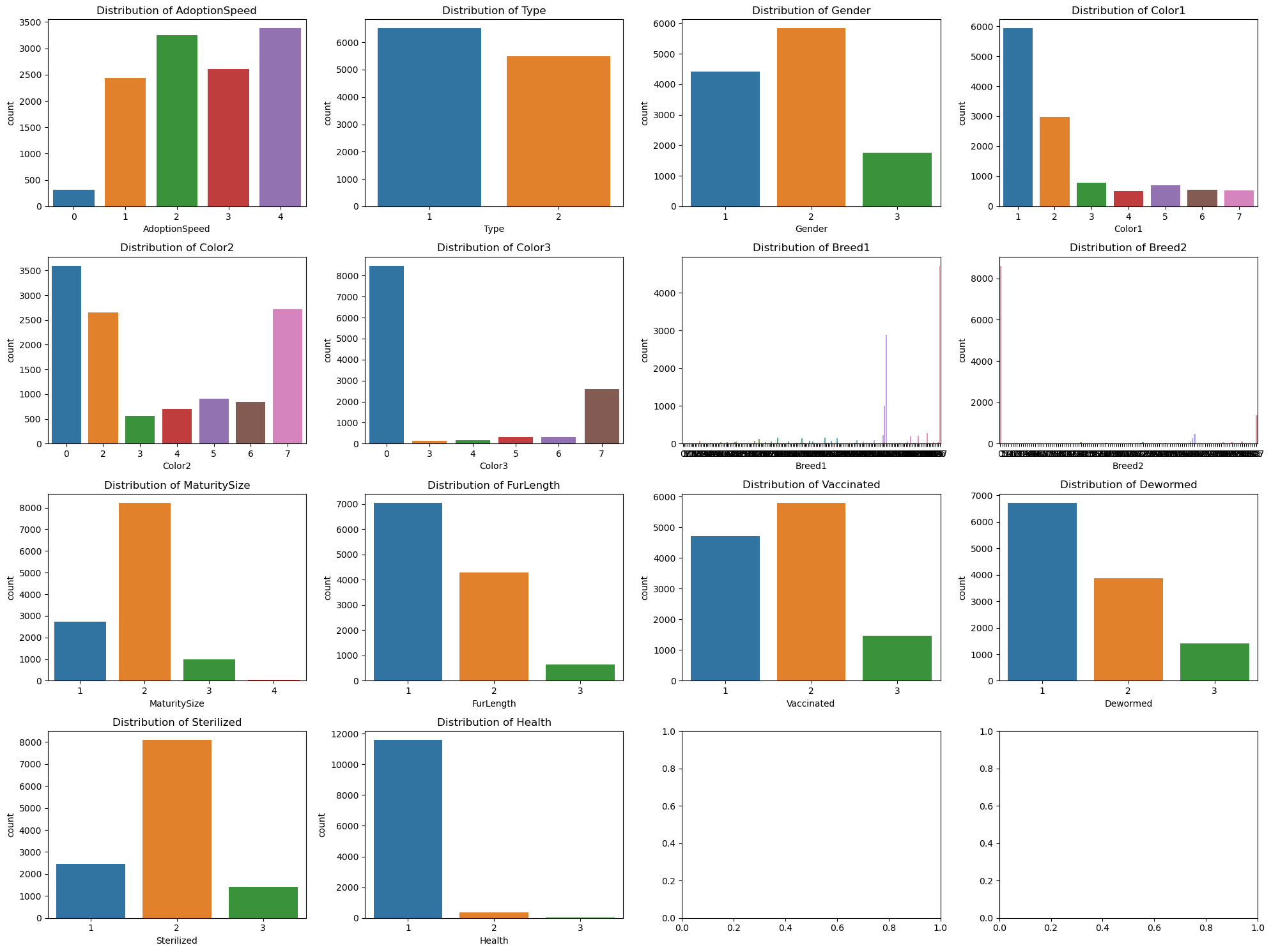
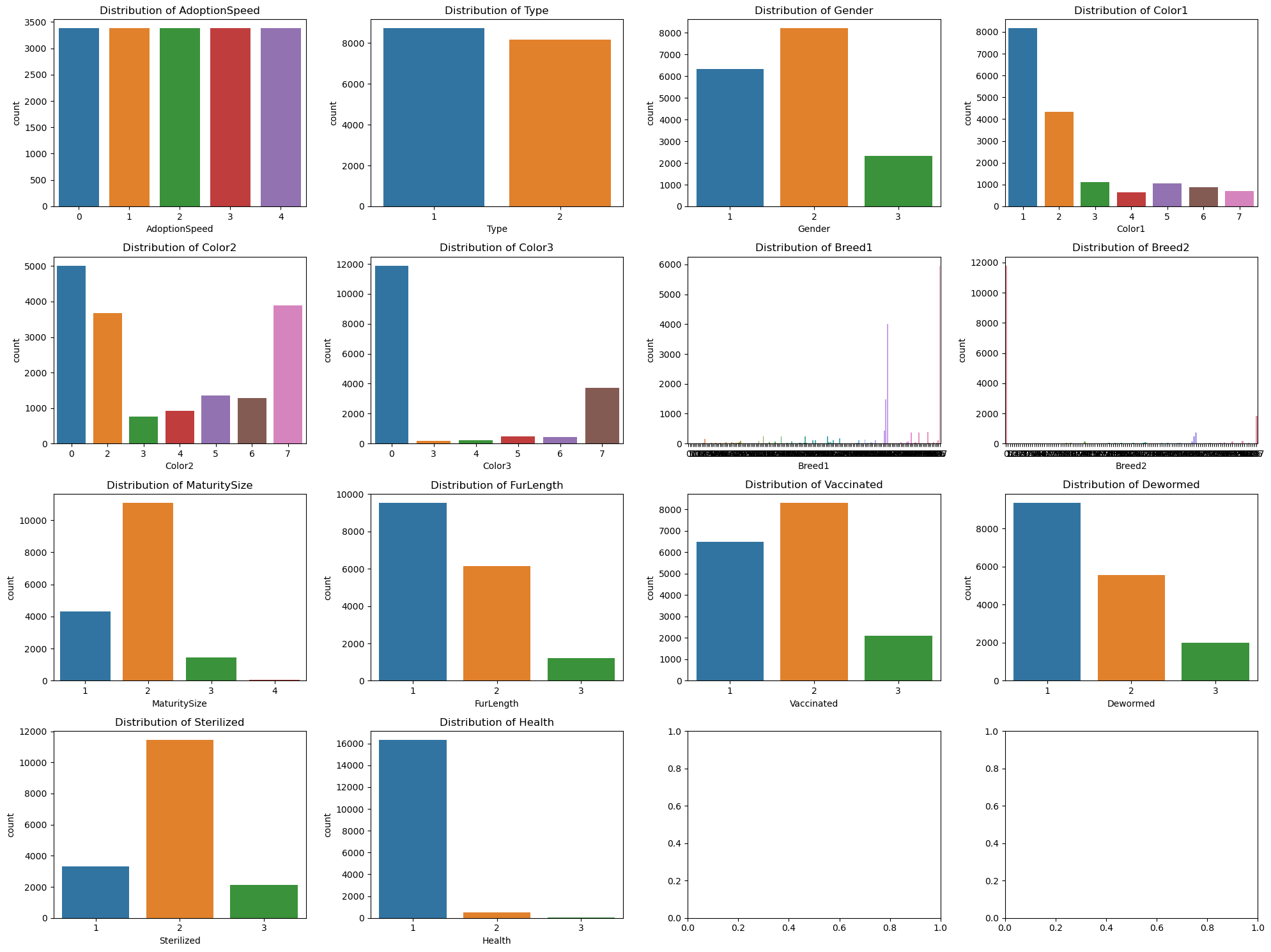
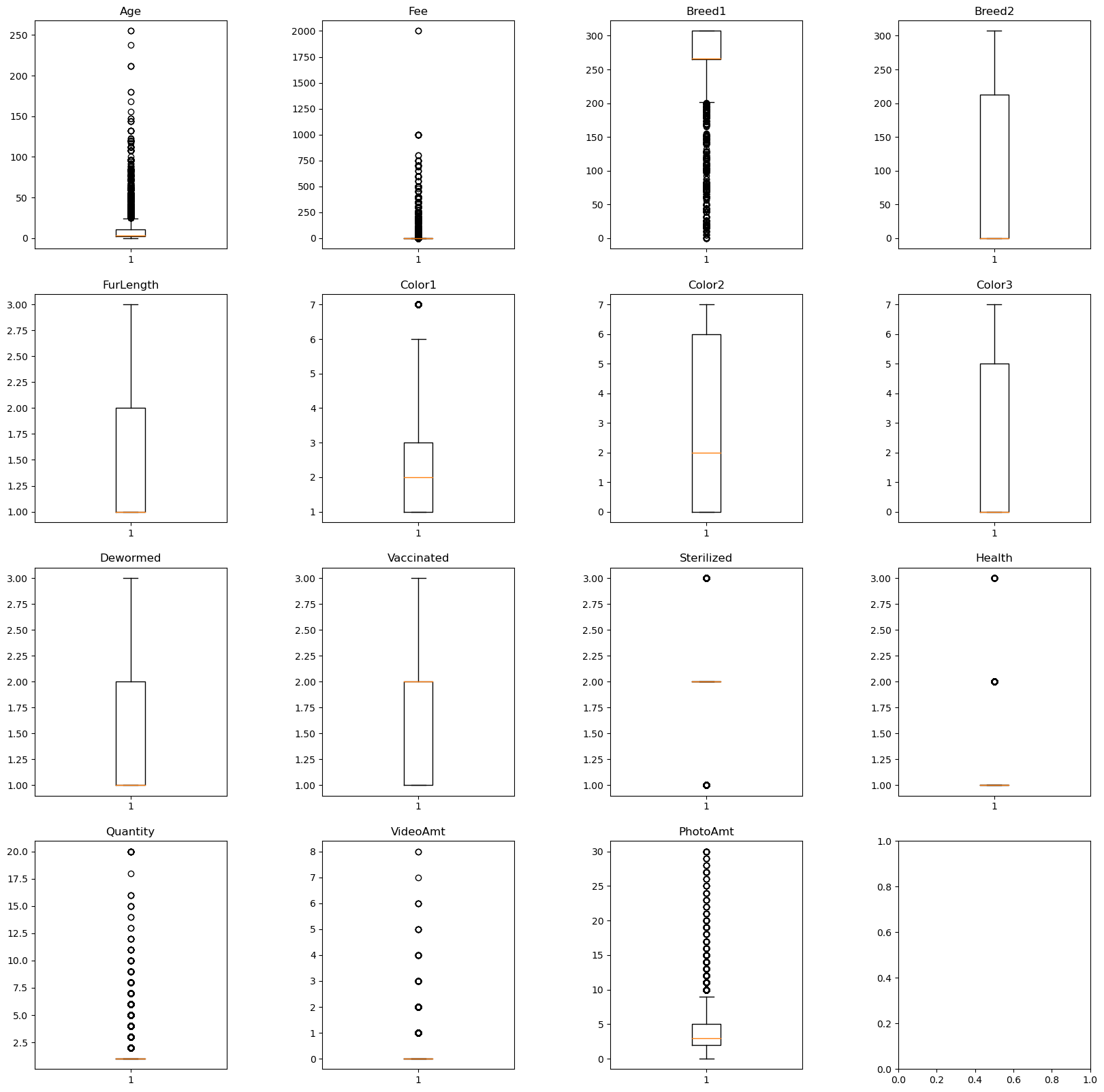
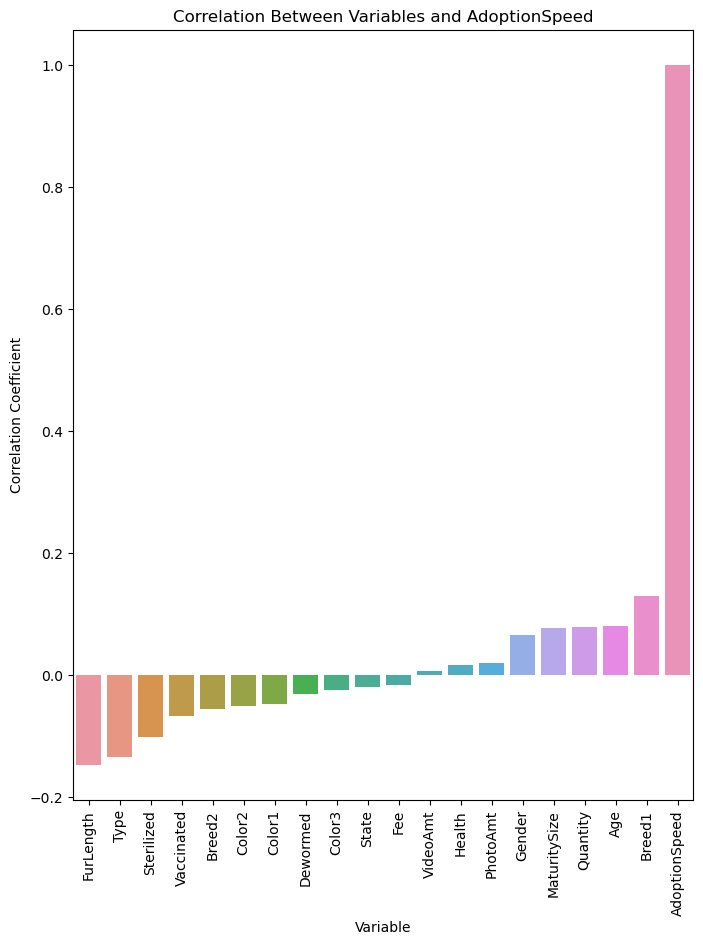
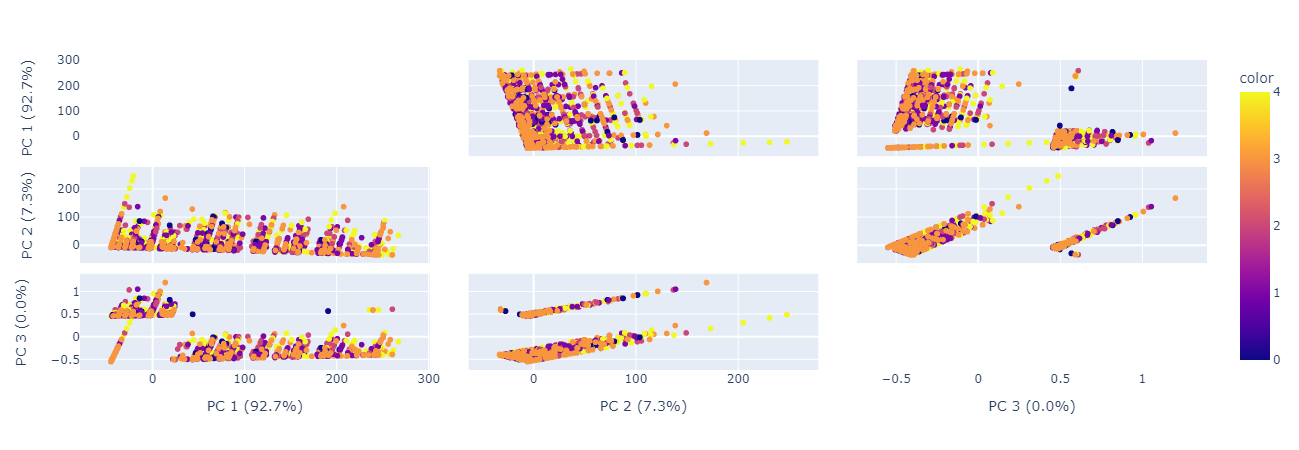
1. Data Cleaning and preprocessing
2. Extracting visualizations and insights from the data after preprocessing
3. Feature Extraction based on the insights (like the correlation with the target variable)
4. Applying PCA to reduce the dimensionality
5. Use spark to create a spark session
6. Read the data (features whether PCA or not and the target variable) into a spark data frame
7. We tried several models with different sets of features:
   1. Some models are used from the MLlib library provided by Spark
   2. Naive Bayes is implemented from scratch by Converting the data frame into RDD, then using a map-reduce algorithm to calculate each needed probability

# 

# 

# Analysis and solution of the problem

## Data Preprocessing

* Split the data into train, and test datasets (80 20)
* Classes need to be balanced (class 0 has a very low count with the perspective of other classes). So, we used the RandomOverSampling method to deal with class imbalance.  
  **Before RandomOverSampling**  
    
    
    
    
    
    
    
    
  **After RandomOverSampling  
  **
* Used boxplot to check for any outlier in all columns to be removed from the dataset or transformed using log transformation techniques to reduce their impact on the model's performance (can cause the model to overfit to the extreme values and result in poor generalization to new data).  
  
* Calculated the correlation between all columns and the target variable after which we selected only the top 3 highest correlated values.  
  
* No need for encoding categorical variables (already done).
* No need for handling missing values (no high percentage of missing values, only 8% in the “name” column which was not used anyways)
* Apply PCA over the most important features and select the top two PCA components  
  

## Data visualization

* We applied multiple visualizations to view the distribution of each category and column in the dataset.
* Due to the use of **RandomOverSampler** to compensate for the data imbalance in the target variable (AdoptionSpeed), we wanted to view the effect over other columns. Is the distribution greatly affected?
* When we received the dataset in the first place, all feature values were encoded ( another file for each feature was provided to explain each code).  
  So, we put extra effort to map the values and make meaningful distributions

### Before Balancing

### 

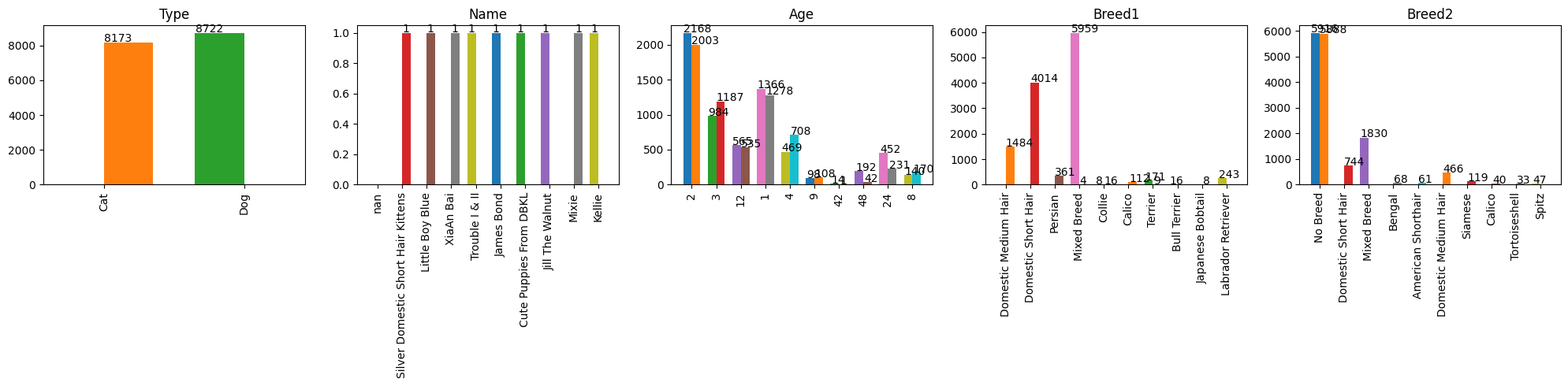
### 

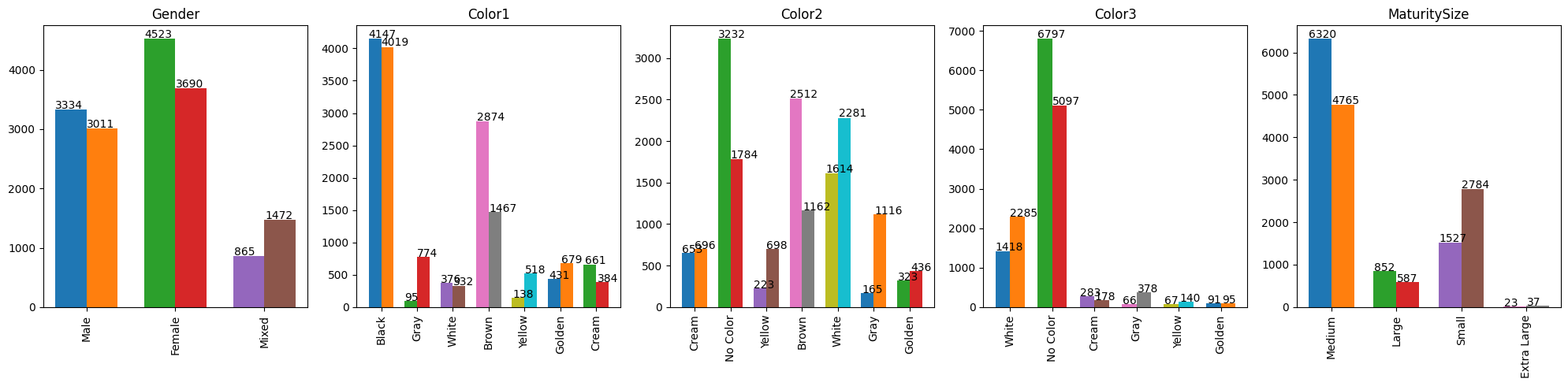
### 

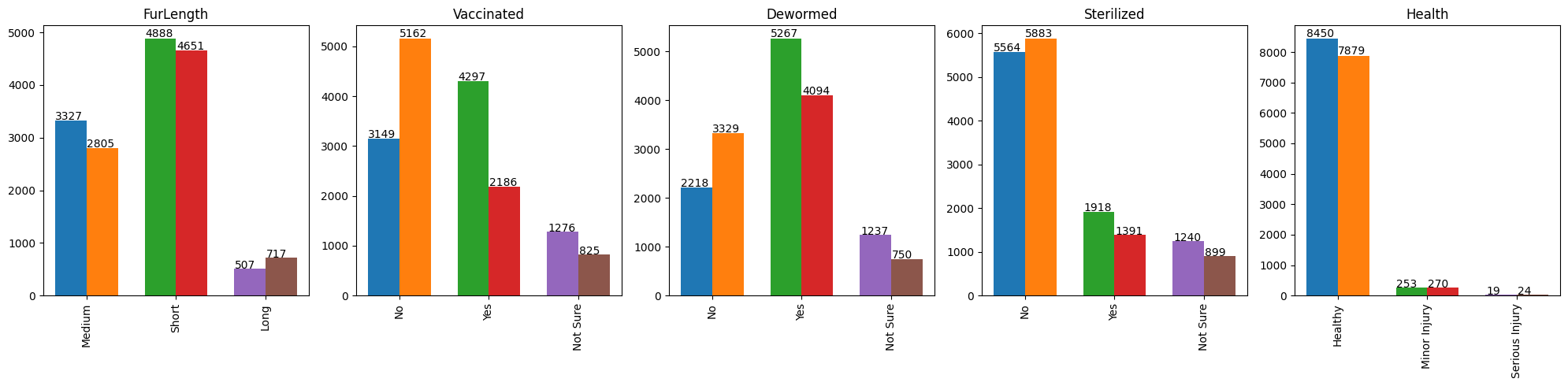
### 

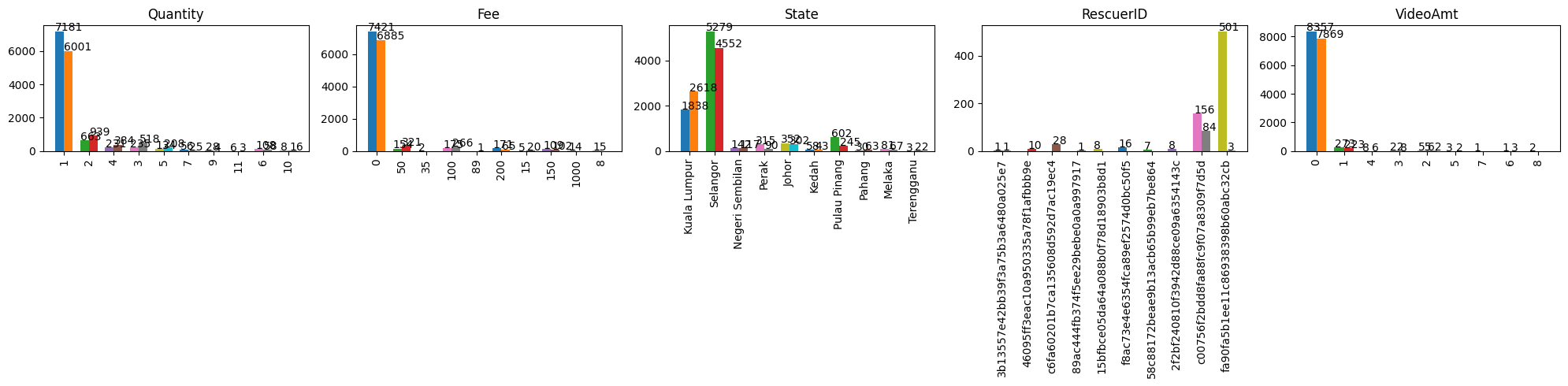
### After Balancing

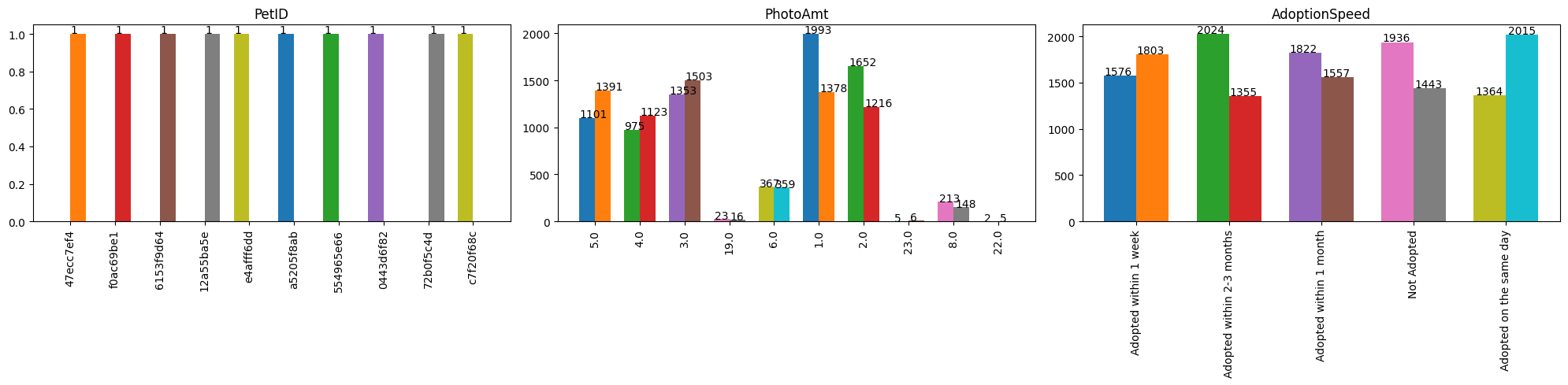
### By looking and comparing the two distributions, the distribution was still the same (just some shift in numbers but keeping the percentage for each value the same) on the features we selected for classification

Also, we wanted to split the distributions on each feature between Cats and Dogs to help us in the Extracting insights phase



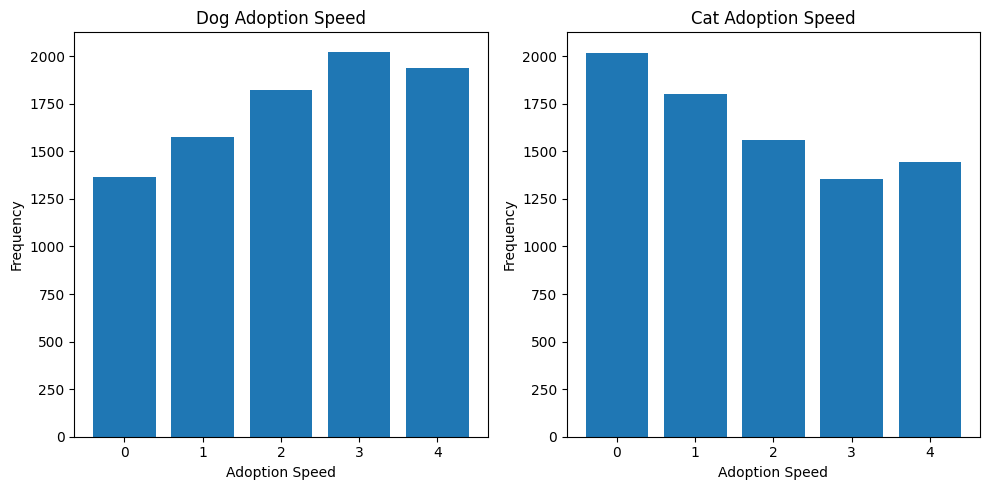
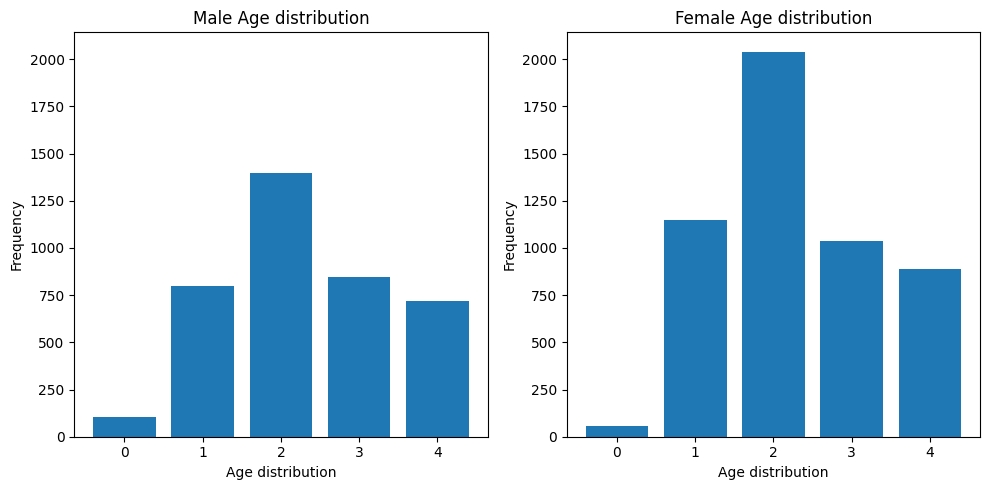
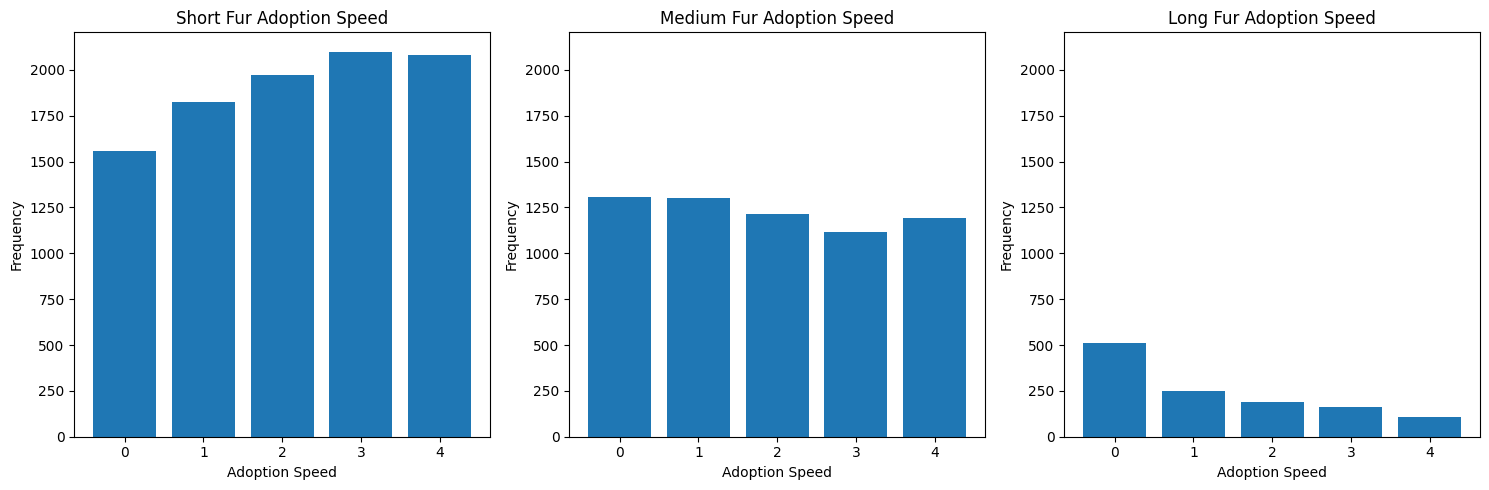
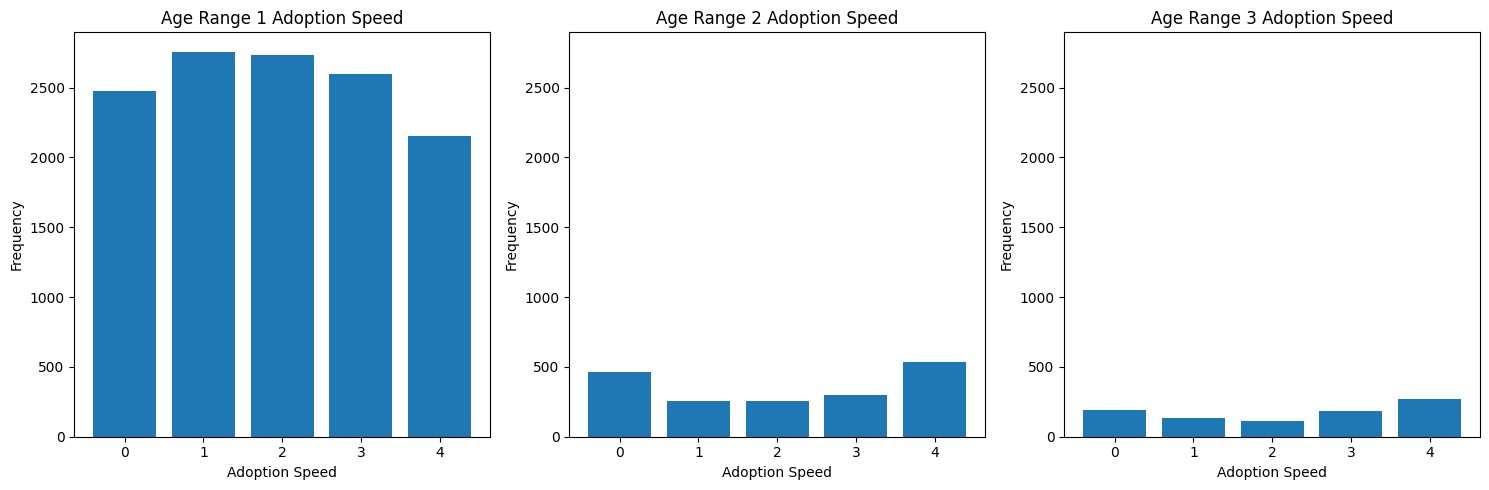
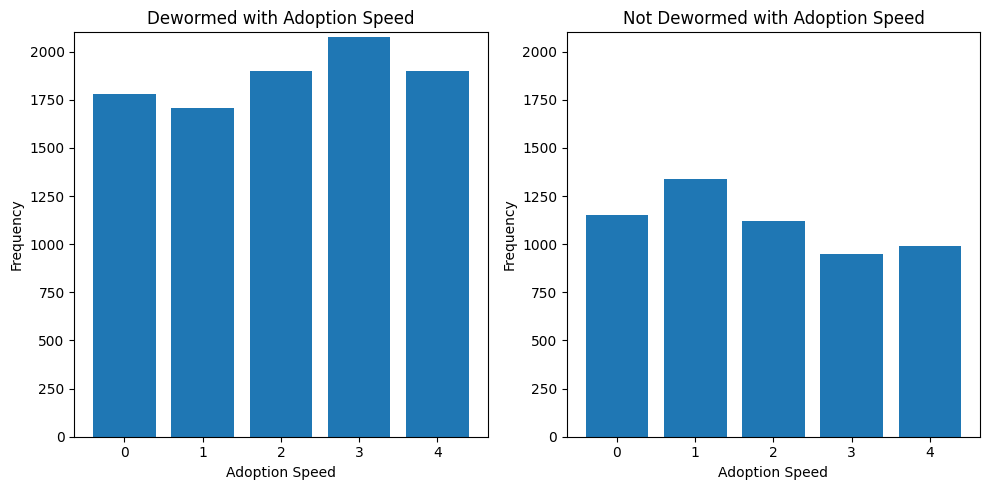
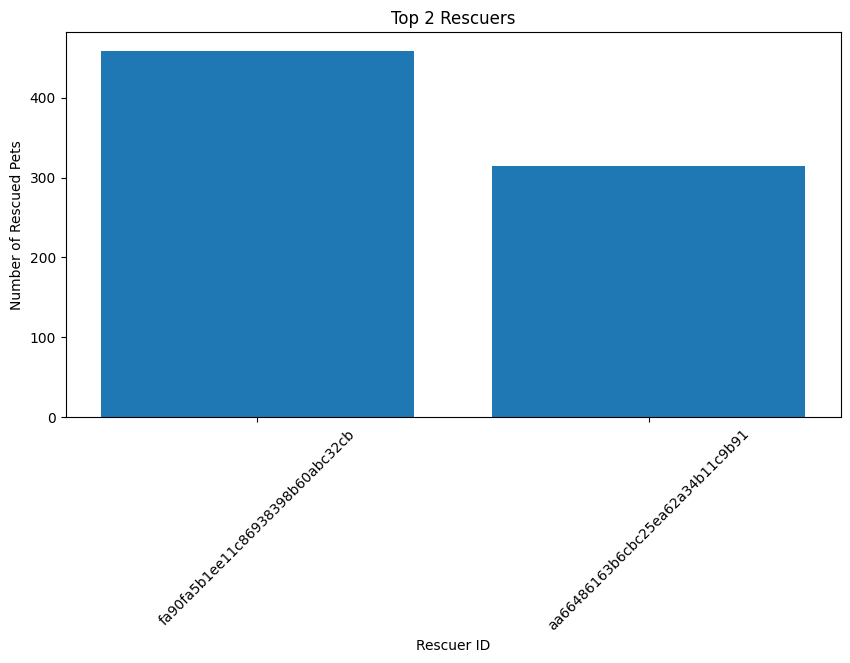






## Extracting insights from data

### During the analysis, we questioned the following

1. Is there a significant difference in the adoption speed between dogs and cats in the dataset?  
   → Yes, on average the number of adopted cats in the fewer durations is more than dogs. However, the number of adopted dogs in longer durations is more than cats.  
   
2. Is there a significant difference in the age distribution between male and female pets in the dataset?   
   → No, it came to our surprise that the age distribution between males and females is the same in the dataset.  
   
3. Does increasing the fur length of a pet increase its chances of adoption?  
   → Yes, Although the pets with long hair are fewer the highest percentage lies in faster adoption speed regions  
   
4. Does the age of a pet have a significant impact on its adoption speed?  
   → We have 3 ranges for age (in months):  
   
   1. 0 - 12: most pets are in this range where they are most frequent at (2) adopted between 8 and 30 days (1st month) after being listed.
   2. 12 - 24: are most frequent at (4) no adoption after 100 days of being listed.
   3. 24 - 36: same as the previous range.
5. Is there a difference in adoption speed whether the pet is vaccinated or not?  
   → Although most of the pets are dewormed, also they are the most adopted  
   
6. Get the top 2 rescuers' IDs using **map-reduce** to give them special rewards:  
   

## Association Rules

We used Pyspark to get the frequent items and rules of the association.

Those are the columns we have applied the apriori on them:

["Type", "Color1", "MaturitySize", "FurLength", "Vaccinated", "Dewormed", "Sterilized", "Health", "AdoptionSpeed"]

* We applied the following process to the dataset before preprocessing
* First, we mapped all the numerical values to non-numeric(meaningful) values
* Convert values of each record to be in the shape: ["id", "items"]
* Use minSupport=0.5, minConfidence=0.6
* Train the model using the FP Growth algorithm which is used to find the frequent itemsets and association rules in a dataset while being faster than the Apriori algorithm.

Results

Frequent Items

Top 11 frequent values in the records:

* 10077 pets are “Not Sterilized”.
* 9782 pets are “Not Sterilized” and “Healthy”.
* 8397 pets are “Dewormed”.
* 8161 pets are "Healthy" and have been "Dewormed".
* 14478 pets are "Healthy".
* 8132 pets are "Dogs".
* 7845 pets are "Healthy" and are "Dogs".
* 10305 pets are mature at "Medium" size.
* 10030 pets are "Healthy" and mature at "Medium" size.
* 8808 pets have "Short Fur".
* 8536 pets have "Short Fur" and are "Healthy".

Association Rules

We have observed the top 7 rules with the highest confidence, left, and support.

Terminologies:

* The confidence indicates the percentage of times that the consequent occurs given the antecedent.
* The lift indicates the strength of the association between the antecedent and consequent, where a lift of 1 indicates no association, a lift greater than 1 indicates a positive association and a lift less than 1 indicates a negative association.
* The support of 0.6524 indicates the percentage of transactions in the dataset that contain both the antecedent and consequent.

Rules:

1. If a pet is not sterilized, then it is likely to be healthy, with a confidence of 0.9707, a lift of 1.0053, and a support of 0.6524.
2. If a pet has short fur, then it is likely to be healthy, with a confidence of 0.9691, a lift of 1.0036, and a support of 0.5693.
3. If a pet is dewormed, then it is likely to be healthy, with a confidence of 0.9719, a lift of 1.0065, and a support of 0.5443.
4. If a pet is healthy, then it is likely to not be sterilized, with a confidence of 0.6756, a lift of 1.0053, and a support of 0.6524.
5. If a pet is healthy, then it is likely to have a medium maturity size, with confidence of 0.6928, a lift of 1.0079, and a support of 0.6690.
6. If a pet is a dog, then it is likely to be healthy, with a confidence of 0.9647, a lift of 0.9990, and a support of 0.5232.
7. If a pet has a medium maturity size, then it is likely to be healthy, with a confidence of 0.9733, a lift of 1.0079, and a support of 0.6690.

Prediction Rules:

If we have some values, we can predict other values.

Example: items=['Cat', 'Cream', 'Large MaturitySize', 'Medium Fur', 'No Vaccinated', 'Yes Dewormed', 'Not Sure Sterilized', 'Healthy', 'Adoption within the first 3 months'], prediction=['No Sterilized', 'Medium MaturitySize']).

## 

## Model/Classifier training

### We implemented 4 different ML models to train over the different sets of features.

* The four models are Logistic Regression, Decision Tree, Random Forest, and Naive Bayes
* All of the models are implemented in Python using the Spark framework
* Each model is trained three times
  + On the features (Type, Age, Breed1) from the training dataset without any preprocessing or balancing for the classes
  + On the same set of features (Type, Age, Breed1) but after the preprocessing steps and class balancing that we explained before
  + On the training dataset after performing PCA and selecting the highest two components (Naive Bayes was not trained on this)

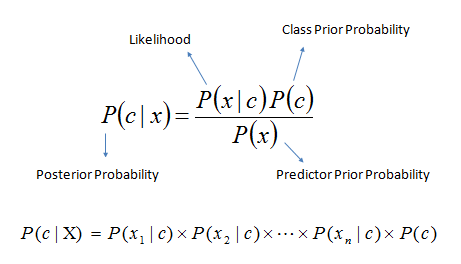
### All models were trained using the Pyspark framework to support multiple processes running concurrently and here are common steps:

1. First, create the SparkSession with the number of threads set to the number available on your machine
2. Read the data into a spark data frame
3. Select only the columns we’re interested in for training
4. Optional: apply some cleaning steps to handle missing values (this was done in the preprocessing steps but done anyway just in case)

**MLlib** is a library by **Pyspark** containing a set of machine learning algorithms for scalability plus tools for feature selection and building ML pipelines for prediction problems.

Before importing and training the classifiers we have to collect the features together in a single column where each value is a vector containing all features.  
→ This was done using the **VectorAssembler** class in the library.

Each one of the four models was imported and used by the **MLlib** library except for the **Naive Bayes** model was implemented from scratch using the map-reduce technique



1. Convert the data frame as an **RDD** to be able to apply the map and reduce functions.
2. Map to count the number of occurrences of each class and each value of the features. Each map contains the value and 1 next to it
3. Reduce the data to count the number of each class using the function **reduceByKey** where the key is the value of the feature. The output will be the unique key and the summation next to it
4. Now we want to count the probability of each class for each value of the features  
   Map the data to count the occurrences of each class with the different values of the features
5. Reduce the data to count the number of each class. The output will be a tuple (the\_feature\_value, the class) and a value is the count of all occurrences of that tuple
6. Convert the schema ((feature, class), count) -> (feature, (class, count))
7. Join the RDDs to get the total number of occurrences of each feature (feature, ((class, count), total)), that's why we needed the above step to convert the schema
8. Convert the schema again (feature, ((class, count), total)) -> (feature, (class, count), total) to be able to divide them easily (OPTIONAL step)
9. Divide the feature count with each class over the total count of the feature
10. Convert the RDD output to a dictionary where each key is the feature value and the value is a list containing the probability of each class (the list is sorted to be able to index each class probability easily)
11. For the classification process, it’s straightforward to calculate the probabilities and multiply them together. Select the class with the highest probability

## Cloud Deployment

Databricks is one of the leading platforms to work with intensive data applications. It enables native integrations with different cloud providers like Azure, AWS, and GCP.

It is worth mentioning that Databricks at its core works above Apache Spark ( same creators on both projects ) and it also supports using notebooks so our code worked smoothly without many modifications and we focused on configuring the cloud environment this way:

1. Go to <https://azure.microsoft.com/en-us/products/databricks> and sign in with our Student Subscription and Create a Databricks workspace.

2. By launching this workspace we’re forwarded to the Databricks dashboard with multiple options to work on the data   
A screenshot of a computer

Description automatically generated

3. We started by creating the computing cluster (only one node with 4 cores due to the student subscription limited quota)

4. Upload both our data and code (notebooks) and run them to get the results.

# 

# Results and Evaluation

For each classifier trained we evaluated using the confusion matrix and the F1 score and here is how it goes:

## On the features (Type, Age, Breed1) from the training dataset without any preprocessing or balancing for the classes

### Logistic Regression: F1 score= 0.245

### Decision Tree: F1 score= 0.296

### Random Forest: F1 score= 0.298

### Naive Bayes: F1 score= 0.201

## On the same set of features (Type, Age, Breed1) but after the preprocessing steps and class balancing that we explained before

### Logistic Regression: F1 score= 0.193

### Decision Tree: F1 score= 0.297

### Random Forrest: F1 score= 0.295

### Naive Bayes (using MLlib): F1 score= 0.172

### Naive Bayes(using Map-Reduce): F1 score= 0.17

## On the training dataset after performing PCA and selecting the highest two components (Naive Bayes was not trained on this)

### Logistic Regression: F1 score= 0.218

### Decision Tree: F1 score= 0.123

### Random Forest: F1 score= 0.208