Data Exploration

Exploring **outcome_type**, we determined there was a class imbalance. One thing to note is that outcome types differed greatly for cats vs dogs. We would use pd.group_by() to pair features with **outcome_type** to better understand how that feature affected that record's outcome.

Starting with **names**, we observed different data distributions with missing names vs names given, animals that were named by the shelter (had a * by them), and the number of words in a name. We explored where animals were found in **found_location** by looking at the city name and if an animal was found on a highway. When looking at **intake_type**, it was found that ~75% of animals were Strays with Owner Surrender as the 2nd most common intake with ~20% of the pets. When looking at the **intake_condition**, we found that ~85% of the animals came in as Normal.

animal_type had a roughly even distribution and **sex_upon_intake** had similar gender distributions, regardless of being intact or spayed/neutered. **age_upon_intake** was written as a string, in the format <number> <unit of time>. Most records had an age upon intake between 0 - 2 years. **breed** was the trickiest column to explore, because there were thousands of unique values. About half of the dataset had a breed that only appeared in 1% or less of all other records. One cause of this is the fact that mixed breeds are differentiated from purebreds, either by adding "Mix" to the end of the breed or adding a slash between two breeds (e.g. Pitbull / Labrador).

For **color**, we noticed darker color animals generally spent longer in the shelter. We also noted that common colors (brown, black, white, gray) dominated the dataset. **intake_time** was largely in the afternoon (between hours 10-18 of the day).

Data Cleaning

First, there was one record in the dataset that was a test record, which we dropped (shown by us dropping the one record that had a NaN age). There was a singular record with an **intake_type** of Wildlife which was also dropped. One record was missing a value for **sex_upon_intake**, which we filled in with "Unknown." Besides that, any columns we kept were not missing values.

Intake_time was formatted in MM/DD/YYYY HH:Mm, which is unhelpful for training a machine learning model. Instead, we converted the string formatted timestamps to UNIX timestamp, which is numerical. Later, we encoded it into multiple features between hour, month, and year. We also created functions to determine if bucketing months into seasons and bucketing hours into times of day would help with certain models.

Since there were so many values for **intake_condition**, we looked at the distributions of intake conditions with **outcome_type** and combined the ones that made sense with common knowledge (ex: nursing and neonatal, congenital and sick) and combined intake condition values that had similar distribution of outcome types (ex: normal and behavior).

age_upon_intake was first converted from categorical to numerical, using years as units. Later, we used months as the unit because of the high imbalance in the distribution when the unit was in years. For records that had negative age values, which we replaced with 0. Since the data was skewed toward younger ages, we also tried using the log() of **age_upon_intake** as a feature.

To reduce **breed's** fragmentation, we first attempted to make an **is_mix** and removed the word "Mix" from all values. We later attempted to separate out breeds with a slash (/), until later feature engineering efforts pushed us to get rid of breed and form breed group clusters using info found from a different Kaggle set about dog breeds and their characteristics. The additional traits in the dataset ended up not improving accuracy.

To clean **color**, we combined conventional ones to reduce dimensionality when one hot encoding. For example, gray is effectively synonymous with blue and silver. We also tried to convert the color into black and non-black (based on discoveries data exploration) and ran it on a few of our best models just to see if it would change anything.

Feature Engineering

After data exploration, we decided to drop **id**, **name**, **outcome_time** because they were not in the test set, meaning that we also were not going to look at days_in_shelter, which also got dropped. We also decided to drop **found_location** and **date_of_birth**, because we felt like other features were more important and we wanted to be wary of the curse of dimensionality.

For **breed,** we first experimented with a **top_10_breed** column, which would be a 1 if a record had a breed that was in the top 10 most common. The distribution in outcome type for cats and dogs seemed similar, so we instead

made an **is_mix** that would be 1 if a record contained “Mix” and a 0 otherwise, and frequency encoded breed. Later on, we used a dataset that categorized breeds by dog group to replace breed with a breed group. We used the elbow method, along with plotting silhouette scores, in order to estimate an ideal k of 9. After mapping all the dogs in our dataset to an external dogs_cleaned.csv, we were able to assign breed to a cluster group instead, drastically decreasing dimensionality.

At first, **color** was frequency encoded without initial adjustments, effectively comparing the commonality of colors. In order to treat it more like a categorical variable, we created a “primary_color” column that was just the first color before any slashes. This reduced the variance enough to one hot encode for models affected by the curse of dimensionality.

Because the values for the numerical version of **intake_time** was high, we encoded intake time into **intake_hour, intake_month,** and **intake_year.** We took this further by splitting time into time of day (morning, afternoon, evening) and season of the year (spring, summer, fall, winter).

Modeling Approach

Decision Tree: After our initial attempts at data cleaning and feature engineering, we decided to use a Decision Tree classifier as our first model. This model used GridSearchCV, max_depth, max_features, and min_samples_leaf as hyperparameters, and we decided to train separate classifiers for cats and dogs and combine their predictions at the end. However, this approach took a very long time to train, for very poor results.

XGBoost: After our poor Decision Tree results, a TA suggested we use an tree ensemble classifier, XGBoost. For our hyperparameters max_depth, learning_rate, n_estimators, subsample, and colsample_bytree. This still took a considerable amount of time, so we downsampled the dataset to 25,000 records, and got a 31% accuracy. We then started using RandomizedSearchCV to cut down on training time, and were able to train the whole dataset, achieving a 37% accuracy.

In later iterations, we attempted to use SMOTE and PCA to mitigate class imbalance and the curse of dimensionality. However, SMOTE increased training times considerably, and PCA (with variance set at 95%) dropped all columns except one, which made us abandon those techniques. We stopped training separate cat and dog classifiers because it essentially doubled the training time, and we started using balanced accuracy along with StratifiedKFolds to align the model with the competition’s scoring metric. We also added new hyperparameters to the map: gamma, reg_alpha, and reg_lambda. Lastly, weighting classification by presence in the dataset helped us address the class imbalance. With all of these techniques in place, we achieved a score of 51.9% in 20 iterations, and a score of 52.3% in 1000 iterations (our highest score). In our second submission, we changed the hyperparameter map to select values from a uniform distribution, either as a float or an int depending on the parameter.

Naive Bayes: After looking at how different features had different distribution of outcome types, we wanted to try using a Multinomial Naive Bayes model, since the multinomial model handled both categorical and numerical data. However, when this model encounters numbers, it expects the numbers to indicate a count or frequency (which did not match what the numbers for intake time or age were meant to mean). To deal with the class imbalance, we wanted to use SMOTE. Since it oversamples the minority class, it helped the model discover patterns for the less represented outcome types, which gave us a score of 47%. Unfortunately, we realized later on that there was data leakage: we had used SMOTE on the dataset before putting it in the pipeline. After we fixed it, our accuracy never exceeded 35%.

We wanted to try our features with Categorical Naive Bayes, so we binned age_upon_intake time specifically since the model could only take categorical data. Running it yielded a low accuracy score of 35%, and we decided to avoid Naive Bayes since our features broke one of the rules: they were not independent from one another (ex: breed and color are correlated with each other and NOT independent).

LinearSVC: Wanting to explore other models, we tested out a LinearSVC model. We one hot encoded the categorical variables, including class weights, and tried running the model with and without using SMOTE. Unfortunately, we forgot to scale the numerical data. Regardless, our balanced accuracy never exceeded 35%. We thought nonlinear SVC models may fit our data better, but we decided to allocate resources towards other models.

CatBoost: After running other models and realizing that categorical variables were a hassle to preprocess, we did some research on the internet and were advised to try CatBoost, a gradient boosting model that handled categorical variables natively. It was extremely convenient to use thanks to this. We also began printing out confusion matrices and realized we were performing poorly on minority classes. Therefore, we began investigating

methods to reduce over predictions of the majority class. At first we tried SMOTE, but this drastically extended runtimes for the same rough accuracies. We tried class weighting and stratified k-folding, which helped significantly with performance on minority classes. Later on, we attempted to increase thresholds for catboost but ran into bugs relating to categorical variables, and decided to explore other models.

We tried ensembling CatBoost to capitalize on our earlier success with it, but the ensembling process took much longer for about the same accuracy. We suspect that the CatBoost we trained were too similar and so were just outputting the same vote for every single record.

Neural Nets: We had heard good things about neural networks from ML students of the last semester. On their advice, we tried a PyTorch MLP. However, it was extremely memory intensive. Rebecca spent a few hours trying to run it on the CS supercomputers, but failed because it didn't accept large jobs from non-researchers. Later, we were able to run it on Casey's home build, but it performed poorly.

At this point we decided to stick to tree-oriented models, since their accuracies were better and they made conceptual sense for categorical data via splitting.

New Pre-Processing: In order to increase our accuracies, we reexamined our preprocessing to lower dimensionality.We decided that frequency encoding was not useful when we wanted to look at a specific breed or color instead of just how common they were. This is when we went back to feature engineering to create cleaner breed names, group together more colors, and make different features out of intake time instead of using the exact time in seconds, as reflected in the Feature Engineering section.

Random Forest / Extra Random Trees: After deciding to try mostly tree-based models that were robust to categorical features, we looked towards random trees. At first we tried a random forest, but after warnings of overfitting from ChatGPT, also trained an extra random trees model. Both of them were exceedingly quick, but random forest performed a bit better. We also started printing their feature importance in order to monitor if our later feature engineering was effective. This was convenient because the random forest was so quick to run. We tried seeing if a Balanced Random Forest, which specifically accounts for class imbalances, would improve scores, but there was minimal improvement.

Trying to take advantage of the speed of the tree models, we tried to train an ensemble of one vs. rest extra random trees classifiers. However, the accuracies were subpar so we opted to continue with random forest instead.

Logistic Regression: We tried to pivot into a Logistic Regression model just to see if it would yield positive results and because of the quick nature it took to run. After preprocessing (one hot encoding the categoricals and using StandardScaler for the numeric features), we tried it with and without SMOTE techniques, both yielding around 47% accuracy. We tried to threshold the predictions to penalize predicting adopted, but the code got so complicated so we decided to focus on ensembling logistic regression with the BaggingClassifier class. Bagging logistic regression models did not improve at all and in some instances made the accuracy worse. We found that this was due to bagging being used to help with high variance models, but logistic regression is already low-variance, so it would not have made much of a difference.

LightGBM: Near the deadline, we converted our XGBoost classifier to Lightgbm, another tree-based ensemble classifier, because we felt like we squeezed out all of the accuracy we could get out of XGBoost. With not as much time to run the model, we were only able to achieve a 51.6% accuracy, which did not beat out our XGBoost's best score.

Stacking: We also wrote a stacking model using our two most successful models: XGBoost and CatBoost. ChatGPT advised us to use logistic regression as the meta model, with out-of-fold stacking. However, this took extremely long (>15 hours) and didn't give a good accuracy.

Voting: Wanting to try out a heterogeneous classifier that didn't have long run times, we tried using the VotingClassifier with Logistic Regression, Extra Random Tree, Random Forest, and XGBoost. We tried to implement a soft-voting system. We tried to implement a soft-voting system. We wanted to implement thresholding where the Adopted class needed to have an average probability of 80%, and we found it was easier to implement this threshold rule by creating a manual voting system that was separate from the VotingClassifier class. Despite the effort, there was a low accuracy of 44%, which was lower than the base classifiers' accuracies.

Multiclass Partitioning - One v. Rest: We trained an ensemble of OVR XGBoosts stacked on a logistic regression model to try to capitalize on the success of our most successful model. We suspect this was overfitted since the accuracies were overly optimistic.