I. Data Preparation & Justification

The first approach I did was attempt to get a brief understanding of what the dataset initially contained. This included going through the file a couple of rows at a time. It became clear that this was a set concerning 2 animal outcomes – adoption and transfer. With each animal outcome, there were various characteristics associated with the particular animal involved. I confirmed the dataset loaded in with a .head() call.

After analyzing both the size and shape of the set, I had found that the set itself was over 131,000 rows with a span of 12 columns, implying a total data count of over 1 million. The next step was deriving the datatypes for each of the columns, as a set this large for a binary prediction seemed redundant from first glance.

Initially, the data was loaded in as all of the type objects. This was incorrect. Another parse of .head() yielded the results of Sex, Outcome Type, Outcome Subtype, Animal Type, Breed and Color should be of type categorical.

Numerical columns should be Age, MonthYear, Date of Birth, DateTime. Before proceeding further, the column Animal ID was dropped. This column provided no new information. Additionally, the 17 duplicated rows found were deleted, and the data was re-indexed.

Missing values were filled in next. Only 3 categories had Null values, including the categories Outcome Subtype, Name and Outcome Type. The Name column was analyzed to contain no useful information, and was dropped. Outcome Subtype missing values were imputed by using the group mode from the Outcome Type. This is because Outcome Type appeared to be closely related to Outcome Subtype, and thus was a great predictor! Only 40 rows had neither Outcome Type or Outcome Subtype, and these rows were dropped. They only constituted a total of 0.03% of the entire dataset regardless.

Once the data was cleaned of NaN values, the next step was to perform the type conversions. The 6 categorical data filters as discussed previously were immediately converted into type category.

The numerical columns were less trivial. All surround either an Age or a Timestamp value. Date of Birth was dropped as a column due to not only relevance, but the fact that Age upon Outcome could directly derive the same value. Instead of keeping both MonthYear and DateTime, I opted to derive the Year and Month as new numerical features for the model. I was able to convert the MonthYear column into the proper categories. Once finished, MonthYear and DateTime were dropped. Lastly, Age upon Outcome was standardized by days. This took the String value and made it a quantifiable integer that gave a proper scaling to how old the animals were at time of observation.

For each of the numerical columns, univariate analysis was recorded in the form of box and histograms, while the categorial columns were analyzed using count plots. The markdown section entitled “Univariate Analysis” goes into further detail.

II. Procedure

Now that the data was cleaned up, the categorical types needed to be converted into boolean inputs to be used in the KNN and SGD Models. The choice of conversion was one-hot encoding, which revealed additional information:

- There were thousands of choices of breeds. This would equate to thousands of boolean flags for each row, which made it clear to drop this column.

- The same was true for colors. Colors did not need to be encoded as there were not only too many categories, but color is not a relevant tool. A dog or cat of vastly different characteristics could be both the same color.
- Outcome Subtype was subsequently dropped. This was for a special reason to be discussed.

Because we are predicting on Outcome Type, only the Animal Type and Sex were encoded. As each had only 4 and 5 categories respectively, this encoded worked out perfectly.

Before training any model, 20% of the data was defined as test data, while 80% was defined as training data. Because the data set is on the order of 100,000+ entries, this gives us a good amount of testing data. A smaller dataset might benefit from 25%-40% testing data to ensure proper validation, while a larger dataset of 1,000,000+ might only need 1%-5% of the set to test on.

The first KNN was a simple KNN with k set to 3. The data was stratified to ensure proportionality and a random seed of 10 was given to reproduce these results.

The second KNN optimized on the k parameter using GridSearchCV. The parameter range was given from 3-13 across 5 CV folds. After fitting and validating, the optimal k value was found to be 11.

Lastly, an SGD Classifier was used to perform linear classification. It was generated with a loss function of logistical loss to capture binary classification in a probabilistic fashion. An alpha of 0.05 was used as discussed in lecture.

III. Performance

The accuracy of each model was 82.9%, 84.5% and 80.4%, respectively. This would imply that a KNN with k=11 is the best choice of these three for raw accuracy scores. However, it's imperative to measure other metrics including precision: [82%, 81%, 82%], recall: [83%, 85%, 80%] and F1: [83%, 84%, 79%].
Each of these are weighted from the classification tables. For this particular problem, recall is the most important metric. Conceptually, not classifying a transfer or adoption is not as practically negative as missing that an animal should be adopted. Higher recall would ensure that we would catch as many adoptions and transfers as possible. For overall recall, the KNN with k=11 still performed the best.

However, each model struggled with transfer outcome recalls. This is important as the models generally performed around the 80% margin in every other category. For specific transfer recalls, the first KNN actually led the other optimized KNN from 71% to 67%. Optimizing GridSearchCV for recall scoring caused accuracy loss when running the model, but this loss might be worth it if transfer recalls are worth more than general classification.

An important sidenote that should be mentioned is the Outcome Subtype column. When using the Outcome Subtype to predict on the Outcome Type, the overall results were always skewed around 99% for each model, regardless of the optimizations. This implied that Subtype was a perfect predictor of Outcome Type, and so for the sake of trivialness, was dropped during the One-Hot encoding stage.

For confidence purposes, I am moderately confident in general performance. With 84% accuracy, this is not production ready, but the model generalizes quite well on a small number of useful categories. A high F1, recall and precision scores tell me that overall it does a sufficient job. However, for niche outcomes like transfers (which occur at the rate of 1:2 compared to adoptions), I would not be confident. The lower recall on transfer values implies that the model would probably underperform on a real dataset on true transfer cases.