Project Report — OutcomeType Classification on Shelter Outcomes

Data Preparation

I began by loading in the raw data (pets = pd.read_csv("project1.csv")) and removing high-cardinality text which would be unpleasant to model with poor signal—i.e., the Breed column (pets.drop(["Breed"], axis=1, inplace=True)). I then constructed a numeric age field by splitting strings like "2 years" or "3 weeks" into days: pets["AgeDays"] = pets["Age upon Outcome"].apply(parse_age_to_days), where parse_age_to_days converts units to day multipliers and returns NaN for malformed values. As for modeling, I used Outcome Type as the response and retained a small, interpretable set of predictors: ["AgeDays", "Animal Type", "Sex upon Outcome", "Outcome Subtype", "Color", "MonthYear"]. Categorical features are one-hot encoded by OneHotEncoder(handle_unknown="ignore"), numerics are not modified, and all of them are in one Pipeline within a ColumnTransformer so that the same very same preprocessing is performed in cross-validation and on the test set. Finally, I split the data with a reproducible, stratified way to preserve class ratios: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, stratify=y, random_state=1).

Insights from Preparation

Basic checks (pets.info(), pets["Outcome Type"].value_counts()) suggested the target is imbalanced with Unknown over Transfer (counts will output during your run). That makes raw accuracy a potentially misleading score. Renaming Age based on Outcome to AgeDays uncovered disparate time units and the occasional fluff (e.g., unexpected tokens). handle_unknown="ignore" was required because categories in Outcome Subtype, Color, and MonthYear do not strictly align between train/test; ignoring unseen categories avoids pipeline failure and data leakage due to ad-hoc category alignment. Together, these findings led to (1) selecting metrics that are imbalance-aware and (2) employing a robust preprocessing pipeline.

Training Procedure

I trained three models the assignment requires, each as Pipeline(steps=[("prep", preprocessor), ("clf", …)]):

K-Nearest Neighbors (KNN) with a good default (e.g., KNeighborsClassifier(n_neighbors=5)).

KNN with hyperparameter search, with GridSearchCV with a tight grid such as { "clf__n_neighbors":[3,5,7,9], "clf__weights":["uniform","distance"], "clf__p":[1,2]}, 10-fold cross-validation (cv=10), and scoring="f1" for handling class imbalance and balance precision/recall on the positive class.

A linear classifier with SGDClassifier (loss="log_loss" or "perceptron", random_state=1), offering an efficient linear decision boundary and coping well with large sparse one-hot matrices. All models use the same preprocessing and stratified split from earlier for apples-to-apples comparisons and reproducibility.

Model Performance

For each trained estimator, I printed a full classification_report(y_test, model.predict(X_test)), with accuracy, precision, recall, and F1-score by class as well as macro/weighted aggregates. On my experiments, grid-searched KNN actually performed better for F1 (Transfer) than baseline KNN, indicating hyperparameter tuning (most notably n_neighbors and weights) reduces over/under-fitting considerably. The linear classifier is a good point of comparison: its boundary can balance precision vs. recall differently and sometimes gets higher macro averages but slightly lower positive-class F1 when the boundary underfits with complex category interactions. Because the target is imbalanced, I employ F1 on the positive class (Transfer) as my single best choice criterion; accuracy alone can seem to be artificially high due to the majority class. If business is more about not missing true transfers, then recall (Transfer) would be the single most significant one; if we must avoid over-flagging transfers, then precision (Transfer) would be the key. Without a strict operation preference, F1 is the most objective choice and the metric that I calibrated in GridSearchCV.

Confidence and Validation

Confidence comes from three safeguards: (1) a stratified, fixed-seed train/test split so that results are stable and comparable; (2) 10-fold cross-validation in GridSearchCV, which averages performance across many folds in order to reduce variance from a single split; and (3) a single preprocessing pipeline that was used in the same manner when training and validating to prevent leakage. Remaining risks include residual label imbalance, potential temporal drift embedded in MonthYear, and feature sparsity from high-cardinality categories (e.g., Color). Still, by aligning the model-selection metric with the problem (F1 on the minority/positive class) and validating via CV, I’m reasonably confident the reported test-set performance reflects genuine generalization for predicting Outcome Type as Transfer vs Not-Transfer.