In [2]:
```
import cv2
import os
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

In [3]:
```
# Define file paths for training and testing data
cat_train_path = "C:\\Users\\visha\\Desktop\\Final Project\\IMAGE\\dog vs cat\\data
dog_train_path = "C:\\Users\\visha\\Desktop\\Final Project\\IMAGE\\dog vs cat\\data
cat_test_path = "C:\\Users\\visha\\Desktop\\Final Project\\IMAGE\\dog vs cat\\datas
dog_test_path = "C:\\Users\\visha\\Desktop\\Final Project\\IMAGE\\dog vs cat\\datas

# Function to load and preprocess data
def load_and_preprocess_data(image_path, label):
    images = []
    labels = []
    for image_name in os.listdir(image_path):
        if image_name.endswith(".jpg") or image_name.endswith(".png"):
            image = cv2.imread(os.path.join(image_path, image_name))
            if image is not None:
                image = cv2.resize(image, (128, 128))  # Resize to a common size
                image = image / 255.0  # Normalize pixel values to [0, 1]

                # Calculate mean RGB values for the image
                r_mean = np.mean(image[:, :, 0])
                g_mean = np.mean(image[:, :, 1])
                b_mean = np.mean(image[:, :, 2])

                images.append([r_mean, g_mean, b_mean])
                labels.append(label)
    return images, labels

# Load and preprocess training data
cat_train_images, cat_train_labels = load_and_preprocess_data(cat_train_path, 0)  #
dog_train_images, dog_train_labels = load_and_preprocess_data(dog_train_path, 1)  #

# Combine training data
X_train = np.array(cat_train_images + dog_train_images)
y_train = np.array(cat_train_labels + dog_train_labels)

# Load and preprocess testing data
cat_test_images, cat_test_labels = load_and_preprocess_data(cat_test_path, 0)  # 0
dog_test_images, dog_test_labels = load_and_preprocess_data(dog_test_path, 1)  # 1
```

In [4]:
```
# Combine testing data
X_test = np.array(cat_test_images + dog_test_images)
y_test = np.array(cat_test_labels + dog_test_labels)

# Train a logistic regression model
model = LogisticRegression(max_iter=1000)  # You can adjust max_iter based on conve
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Print evaluation metrics
```

```
print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(confusion)
print("Classification Report:")
print(classification_report_str)
```

```
Accuracy: 0.5585
Confusion Matrix:
[[598 402]
 [481 519]]
Classification Report:
              precision    recall  f1-score   support

           0       0.55      0.60      0.58      1000
           1       0.56      0.52      0.54      1000

    accuracy                           0.56      2000
   macro avg       0.56      0.56      0.56      2000
weighted avg       0.56      0.56      0.56      2000
```

In [5]:
```
from sklearn import svm
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Assuming X_train and y_train are your training data and labels
# Similarly, X_test and y_test are your test data and labels

# Create an SVM classifier (you can choose the kernel and other hyperparameters)
clf = svm.SVC(kernel='linear', C=1.0)

# Train the SVM on the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Print evaluation metrics
print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(confusion)
print("Classification Report:")
print(classification_report_str)
```

```
Accuracy: 0.5465
Confusion Matrix:
[[646 354]
 [553 447]]
Classification Report:
              precision    recall  f1-score   support

           0       0.54      0.65      0.59      1000
           1       0.56      0.45      0.50      1000

    accuracy                           0.55      2000
   macro avg       0.55      0.55      0.54      2000
weighted avg       0.55      0.55      0.54      2000
```

In [6]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
# Assuming X_train and y_train are your training data and labels
# Similarly, X_test and y_test are your test data and labels

# Create a Random Forest classifier (you can choose hyperparameters)
clf = RandomForestClassifier(n_estimators=100, max_depth=None, random_state=0)

# Train the Random Forest on the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Print evaluation metrics
print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(confusion)
print("Classification Report:")
print(classification_report_str)
```

```
Accuracy: 0.54
Confusion Matrix:
[[572 428]
 [492 508]]
Classification Report:
              precision    recall  f1-score   support

           0       0.54      0.57      0.55      1000
           1       0.54      0.51      0.52      1000

    accuracy                           0.54      2000
   macro avg       0.54      0.54      0.54      2000
weighted avg       0.54      0.54      0.54      2000
```

In [8]:

```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import make_scorer, accuracy_score

# Create your Random Forest classifier
clf = RandomForestClassifier(n_estimators=100, max_depth=None, random_state=0)

# Define the feature matrix (X_train) and labels (y_train) based on your training d
# X_train contains your feature vectors
# y_train contains your training labels (0 for cats, 1 for dogs)

# Specify the number of folds for cross-validation (e.g., 5-fold)
k = 5

# Create a cross-validation object (KFold)
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Define a custom scorer for accuracy
accuracy_scorer = make_scorer(accuracy_score)

# Perform cross-validation and get accuracy scores
accuracy_scores = cross_val_score(clf, X_train, y_train, cv=kf, scoring=accuracy_sc

# Calculate the mean and standard deviation of accuracy scores
```

```
mean_accuracy = accuracy_scores.mean()
std_accuracy = accuracy_scores.std()

# Print the mean accuracy and standard deviation
print(f"Mean Accuracy: {mean_accuracy}")
print(f"Standard Deviation of Accuracy: {std_accuracy}")
```

```
Mean Accuracy: 0.534
Standard Deviation of Accuracy: 0.013065938542638248
```

In [9]:

```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.svm import SVC
from sklearn.metrics import make_scorer, accuracy_score

# Create an SVM classifier
svm_classifier = SVC(kernel='linear', C=1)  # You can adjust the kernel and C param

# Specify the number of folds for cross-validation (e.g., 5-fold)
k = 5

# Create a cross-validation object (KFold)
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Define a custom scorer for accuracy
accuracy_scorer = make_scorer(accuracy_score)

# Perform cross-validation and get accuracy scores
accuracy_scores = cross_val_score(svm_classifier, X_train, y_train, cv=kf, scoring=

# Calculate the mean and standard deviation of accuracy scores
mean_accuracy = accuracy_scores.mean()
std_accuracy = accuracy_scores.std()

# Print the mean accuracy and standard deviation
print(f"Mean Accuracy: {mean_accuracy}")
print(f"Standard Deviation of Accuracy: {std_accuracy}")
```

```
Mean Accuracy: 0.5418749999999999
Standard Deviation of Accuracy: 0.009834569131385447
```

In [ ]: