Code Document: Pets Facial Expression Classification

# 1. Importing Libraries

The necessary libraries for data manipulation, visualization, and deep learning are imported.   
Libraries like `cv2` for image processing, `seaborn` and `matplotlib` for plotting, and `tensorflow`   
for building deep learning models are included. Additional libraries for missing value analysis, plotting,   
and metric evaluation are also imported.

# 2. Dataset Paths and Data Preprocessing

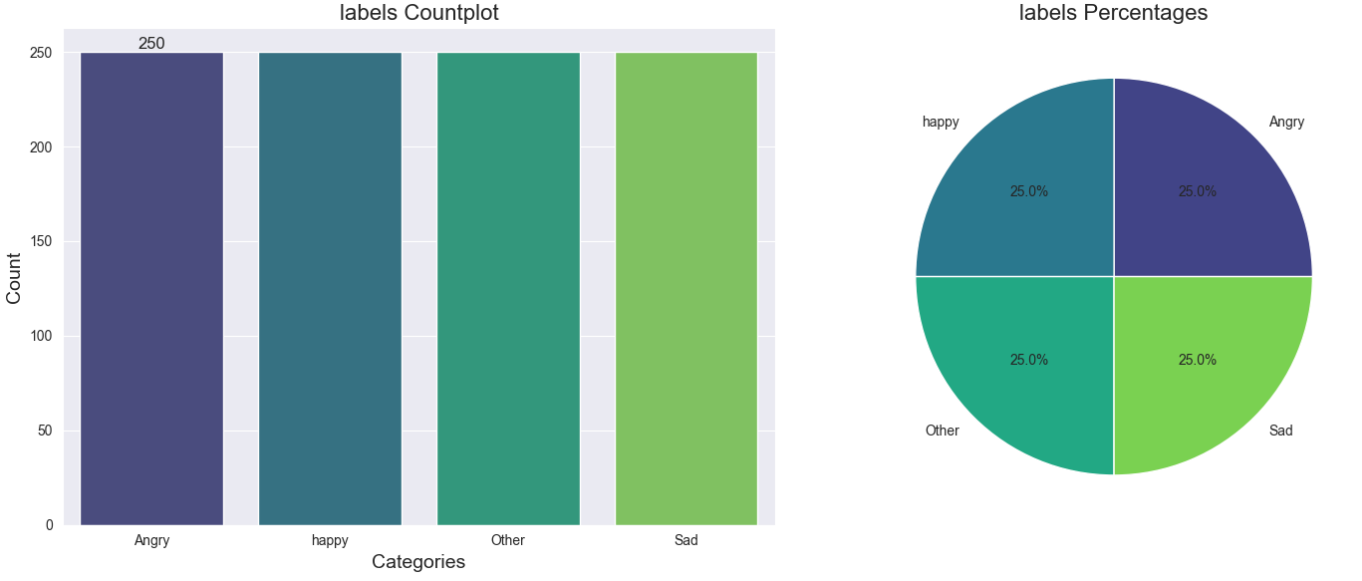
The paths to the training, validation, and test directories are defined here. These paths contain the images   
that will be used for training and evaluation.

# 3. Data Path Generation and Dataframe Creation

The `generate\_data\_paths()` function recursively lists the paths of image files and assigns labels to them   
based on their respective directories. The `filepaths` and `labels` are then returned to be used in further analysis.

# 4. Data Exploration: Labels and Class Distribution

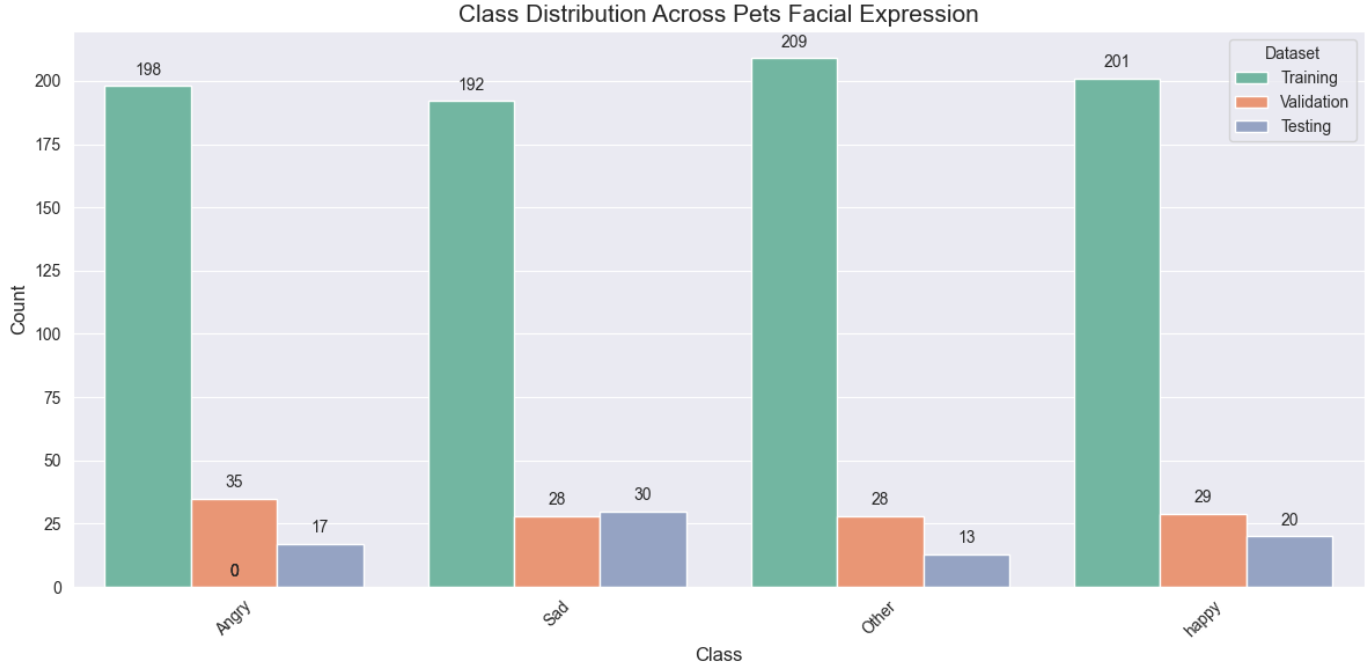
Functions like `num\_of\_examples()`, `num\_of\_classes()`, and `classes\_count()` are used to explore the dataset.   
They print the number of examples (images), the number of unique classes, and the count of images in each class.  
The function `cat\_summary\_with\_graph\_alt()` visualizes the distribution of classes using a count plot (bar plot)   
and a pie chart.



# 5. Splitting the Data

The dataset is split into training, validation, and testing sets. The `train\_df` will hold 80% of the data,   
and the remaining data is divided into `valid\_df` (60% of the remainder) and `test\_df` (40%).

# 6. Class Distribution Visualization

The `plot\_class\_distribution()` function plots the distribution of classes across the training, validation,   
and test datasets using a bar plot and annotations.  


# 7. Image Preprocessing and Data Augmentation

The `ImageDataGenerator` is used for data augmentation, such as rotation, shifting, and flipping, during   
training and testing. The `scalar` function is applied to each image before it is fed to the model.

# 8. Model Creation

An EfficientNetB5 model is loaded as the base model, without the top layer, for feature extraction. The model   
weights are frozen (`trainable = False`) to prevent updates during training. The model is then built using custom   
layers such as `BatchNormalization`, `Dense`, and `Dropout` to prevent overfitting and fine-tune the model for   
classification tasks.

# 9. Compiling the Model

The model is compiled with the Adamax optimizer, categorical cross-entropy loss function (for multi-class   
classification), and accuracy metric.

# 10. Model Training and Early Stopping

The model is trained on the `train\_gen` data for a specified number of epochs. Validation is performed on   
the `valid\_gen` data after each epoch. The `EarlyStopping` callback is used to stop training if validation   
accuracy doesn't improve after 5 epochs, and the best weights are restored.

**11. Training History Plot**

In this section, the training and validation loss and accuracy are plotted for each epoch.  
- `tr\_acc` and `tr\_loss` store the training accuracy and loss values for each epoch.  
- `val\_acc` and `val\_loss` store the validation accuracy and loss values.  
- The epochs are plotted against loss and accuracy for both training and validation.  
- The epochs with the lowest validation loss and highest validation accuracy are highlighted with scatter points.

**12. Model Evaluation**

The model is evaluated on the training, validation, and test datasets. The number of steps for evaluation is   
calculated based on the test dataset size and batch size.  
- The model's performance on each dataset is evaluated using the `evaluate` function.  
- The results are printed showing the loss and accuracy for each dataset.

**13. Model Prediction**

Once the model is trained, it is used to make predictions on the test dataset.  
- `preds` stores the predicted values for the test dataset.  
- `y\_pred` stores the predicted classes, which are extracted by taking the argmax of `preds`.

**14. Confusion Matrix**

A confusion matrix is generated to show the performance of the classification model.  
- The confusion matrix (`cm`) is created using the `confusion\_matrix` function.  
- The matrix is visualized using a heatmap, with annotations showing the values in each cell.  
- The x-axis represents the predicted labels, and the y-axis represents the true labels.

**15. Classification Report**

The `classification\_report` function from `sklearn` generates a detailed report that includes precision, recall,   
f1-score, and support for each class in the test dataset.   
- The `target\_names` are the class labels, which are passed to the function to match with the predicted classes.