1. Implementation Details for Detecting Mislabeled Data:

This report shows the implementation details for detecting mislabeled data in a linear regression model. The approach uses residual analysis to identify potential outliers which may indicate mislabeled data points.

Packages Used:

pandas (imported as pd): For data manipulation and CSV file operations
numpy (imported as np): For numerical computations
sklearn.linear_model: For implementing Linear Regression

Parameter Settings:

Data Loading: The data is loaded from a CSV file named 'linear_regression.csv' using pandas.
Data Preparation: Features(X) are extracted from all columns except the last one.Target variable (y) is taken from the last column.
Model Training: A Linear Regression model is instantiated and fitted on the entire dataset.
Linear Regression: Default parameters are used (no specific hyperparameters set)
Residual Calculation: Predictions are made on the training data. Residuals are calculated as the difference between actual and predicted values.
Outlier Detection: The standard deviation of residuals is computed. Data points with residuals exceeding 3 times the standard deviation are flagged as outliers.
Result: A new column 'Outlier' is added to the dataset, with 1 indicating an outlier and 0 otherwise. The updated dataset is saved to a new CSV file 'linear_regression_done.csv'.

2. Implementation Details for Image Compression using K-means Clustering:

This report shows the implementation details for compressing images using K-means clustering. The approach involves reducing the color palette of one image and applying it to compress both the original and a second image.

Packages Used:

numpy (imported as np): For numerical computations
matplotlib.pyplot (imported as plt): For data visualization
sklearn.cluster: For implementing K-means clustering
sklearn.neighbors: For nearest neighbor computations
PIL (Python Imaging Library): For image loading and manipulation

Parameter Settings:

Data Loading: Two images ('image1.png' and 'image2.png') are loaded using PIL's Image.open() function.
Data Preparation:Images are converted to NumPy arrays using np.array(). Image arrays are reshaped into 2D arrays (pixels1 and pixels2) with dimensions (n_pixels, 3) using reshape(-1, 3).
Optimal k Determination: Plotted the graph for the elbow method to determine the optimal value of K ranging from 1 to 20. From the graph, the **optimal k was around 4**. This means that if we choose a value of 4, we are getting 4 colors from our original image. But, if we want to get more colors from the original image, we can use a higher k value. In the image_edit.py code, I have used k as 15 which gives the image1_done and image2_done a better color(eg. Choosing higher k ensures yellows are retained from the original image).
Model Training: KMeans is instantiated with n_clusters=15 and random_state=42(to generate the same results across different executions of the algorithm)
K-means Clustering: Default parameters are used except for n_clusters and random_state. Centroids and labels are obtained from the fitted model.
Image Compression: For image1: Compressed image is created by replacing each pixel with its corresponding centroid color. For image2: NearestNeighbors is used with n_neighbors=1 to map each pixel to the nearest centroid from image1.
Output: Compressed images are displayed and saved as 'image1_done.png' and 'image2_done.png' in the same folder.