PRML

Programming Assignment - 2

Palash Khatod
B22EE095

Importing the necessary libraries:

In the initial step, essential libraries are imported for data analysis and visualization. Pandas facilitates data manipulation, Matplotlib aids in creating plots, and NumPy supports numerical computations. Scikit-learn's `train_test_split` is employed for dataset partitioning, while Seaborn enhances data visualization. The `r2_score` from Scikit-learn is used for assessing the performance of regression models. These libraries collectively empower the subsequent data exploration and modeling processes.

Question No. - 1

The code performs several tasks related to handling missing values, encoding categorical variables, visualizing feature-target relationships, and implementing a basic decision tree classifier.

Loading Data:

The code reads the Titanic dataset from a CSV file using Pandas.

Handling Missing Values:

Missing values in the 'Age' column are imputed based on conditions related to 'Sex' and 'Pclass'.

Rows with missing values in the 'Embarked' column are dropped.

Dropping Unnecessary Columns:

Columns 'Cabin', 'PassengerId', 'Name', and 'Ticket' are dropped from the dataset.

Encoding Categorical Variables:

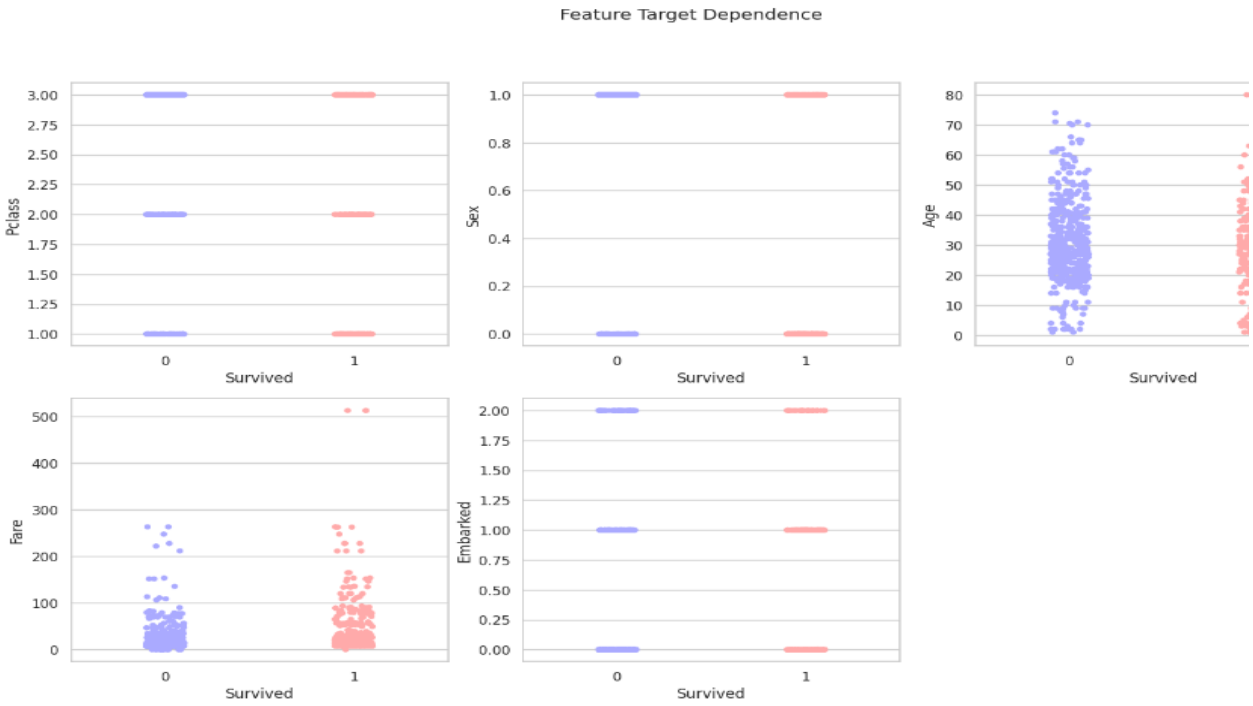
'Sex' column is encoded as 0 for 'female' and 1 for 'male'.

'Embarked' column is encoded as 0 for 'S', 1 for 'C', and 2 for 'Q'.

Visualizing Feature-Target Relationships:

The code creates a subplot with strip plots to visualize the relationships between 'Survived' and other features like 'Pclass', 'Sex', 'Age', 'Fare', and 'Embarked'.

In this dataset, PClass is ORDINAL Dataset Age, Name, PassengerID, Ticket, Fare are Continuous Rest Columns are Nominal Dataset.



Train-Test Split:

Train-Test Split:

The dataset is split into training and testing sets using a manual split with 80% for training and 20% for testing.

Decision Tree Classifier:

Decision Tree Implementation: Functions for building a decision tree (`build_tree`), predicting with the tree (`predict`), and fitting the decision tree (`decision_tree_fit`) are defined.

Decision Tree Training :A decision tree classifier is trained on the training set with specified hyperparameters (`max_depth=3`, `min_samples_leaf=5`).

Decision Tree Prediction:The trained decision tree is used to make predictions on the test set.

Evaluation Metrics:

Confusion Matrix: A confusion matrix is computed to evaluate the performance of the decision tree classifier.

Accuracy Score: The accuracy of the classifier is computed and printed.

Classification Report: Precision, recall, and F1-score are computed for each class ('Survived' and 'Not Survived') and presented in a classification report.

```
Shape of X_train: (694, 7)
Shape of y_train: (694,)
Shape of X_test: (195, 7)
Shape of y_test: (195,)
Predicted values:
[0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0,
Confusion Matrix:
{0: {0: 104, 1: 12}, 1: {0: 16, 1: 63}}
Accuracy: 85.64102564102564

Classification Report:

{'precision': {0: 0.8666666666666667, 1: 0.84}, 'recall': {0:
0.896551724137931, 1: 0.7974683544303798}, 'f1-score': {0:
0.8813559322033899, 1: 0.8181818181818181}, 'support': {0: 116, 1: 79}}
```

Example Usage:

Example Usage:

> The decision tree classifier is created, predictions are made on the test set, and evaluation metrics (confusion matrix, accuracy, and classification report) are printed.

Question No. - 2

Loading the dataset:

The data is loaded from an external source using Pandas, with the dataset retrieved from a specified URL. This step is crucial for subsequent analysis and modeling, allowing seamless access to the TV marketing and sales data required for the simple linear regression model.

Exploratory Data Analysis:

The Exploratory Data Analysis (EDA) in this code involves a concise exploration of the dataset. Displaying the initial rows provides a snapshot of the data, aiding in understanding its structure. The subsequent creation of a scatter plot visually represents the relationship between the TV marketing budget and sales, offering insights into potential patterns or trends. Utilizing Matplotlib and Seaborn libraries, this visualization serves as a foundational step in comprehending the dataset's characteristics, guiding subsequent preprocessing and modeling decisions. The scatter plot effectively conveys the distribution of data points, facilitating a preliminary assessment of the linear relationship between the variables.

In subsequent steps mean and standard deviation is being found and displayed.

```
0  230.1   22.1
1   44.5   10.4
2   17.2    9.3
3  151.5   18.5
4  180.8   12.9
```


```
The mean of TV Marketing Budget is :  147.0425
The mean of Sales is :  14.0225
The Standard Deviation of TV Marketing Budget is :  85.85423631490808
The Standard Deviation of Sales is :  5.217456565710478
```

Data Preprocessing:

Involves essential transformations to enhance model performance.In the provided code, two types of normalization are applied for distinct reasons:

Scaling 'Sales' to [0, 1]: Normalizing the 'Sales' column to a [0, 1] range is beneficial when dealing with variables of different scales. This type of normalization ensures that all values are proportionally represented, preventing dominance by variables with larger magnitudes, and improving convergence during gradient descent in the regression model.

Scaling 'TV' by Dividing by 100: Dividing the 'TV' column by 100 is a form of feature scaling to bring the values to a more manageable scale. This aids in numerical stability during model training and helps prevent potential convergence issues, especially when

using gradient descent, where large input values might result in slower convergence or overshooting of optimal parameters.

Data Splitting:

Data splitting is executed using the train_test_split function, dividing the dataset into training and testing sets. With an 80-20 split, 80% of the data is allocated for training the simple linear regression model, while the remaining 20% serves for assessing its performance.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   TV      200 non-null    float64
 1   Sales   200 non-null    float64
dtypes: float64(2)
memory usage: 3.2 KB
      TV     Sales
0  2.301  0.807087
1  0.445  0.346457
2  0.172  0.303150
3  1.515  0.665354
4  1.808  0.444882
X_train shape: (160,)
X_test shape: (40,)
y_train shape: (160,)
y_test shape: (40,)
```

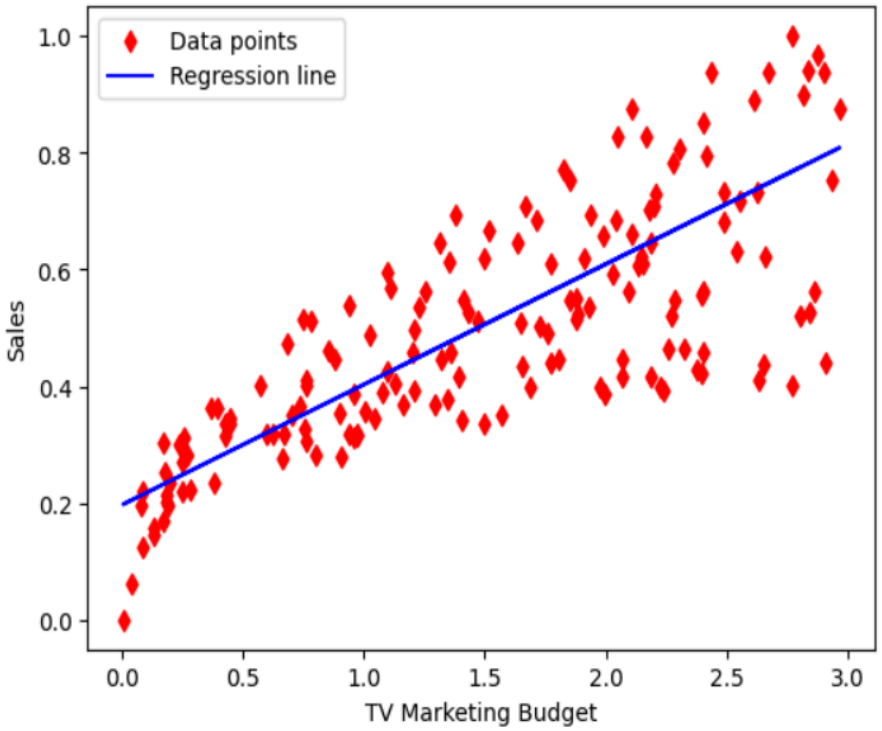
Gradient Descent Function:

The gradient descent function is a pivotal component in optimizing the simple linear regression model. Implemented in the code, it iteratively refines model parameters, slope (m) and intercept (b), to minimize the difference between predicted and actual values. Through 5000 iterations, the function computes the gradients of the cost function with respect to model parameters, adjusting them proportionally. This iterative approach converges towards optimal values. The function is applied separately to the

training and testing sets, facilitating the model's ability to generalize. Its output includes the optimized parameters and the predicted values, essential for assessing model performance.

Plot of Regression line:

The visualization of the regression line involves plotting the training data points and the learned regression line using Matplotlib. This graphical representation enhances understanding by illustrating how well the linear model fits the training data, offering insights into the relationship between TV marketing budget and sales in a clear and interpretable manner.



Model Evaluation:

In the final stage, model evaluation is conducted using key metrics on the test data. Mean Squared Error (MSE) and Mean Absolute Error (MAE) are calculated to quantify the performance of the simple linear regression model.

For my model:

```
mean squared error on test data 0.015372613459096879
Absolute error on test data 0.09224840878882747
```

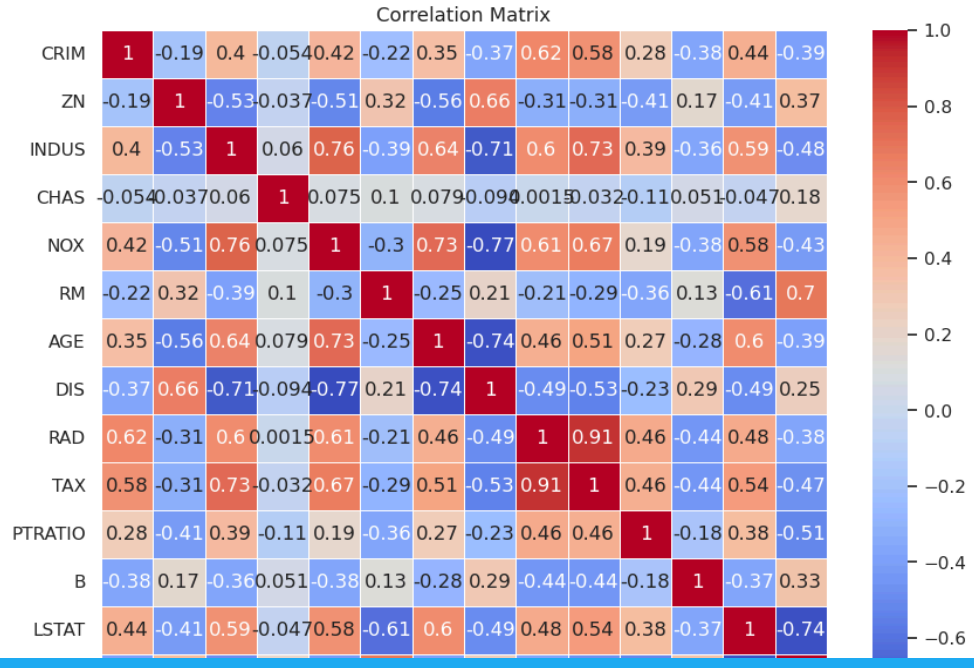
Question No. - 3

1. Loading the Dataset:

The code begins by loading the Boston housing dataset from a CSV file and printing the initial rows to understand the data structure.

2. Correlation Matrix Visualization:

A correlation matrix is computed and visualized using a heatmap, highlighting relationships between different features. This step aids in identifying potential predictors for the target variable, 'MEDV' (Median House Value).



3. Handling Null Values:

Null values are imputed for both float and integer columns using mean and median values, respectively.

4. Normalization:

Feature normalization is applied, scaling each feature to a range between 0 and 1. This ensures uniformity in the data and prevents any feature from dominating the regression process.

5. Initial Visualization of the Dataset:

Scatter plots are created to visualize the relationship between each feature and the target variable, 'MEDV.' This step provides an initial understanding of how individual features relate to the median house value.

6. Data Splitting:

The dataset is split into training and testing sets using the `train_test_split` function, with 80% of the data reserved for training the model.

7. Multivariate Regression Model Training:

A multivariate regression model is implemented using gradient descent. The model is trained on the training set, with weights and bias updated iteratively to minimize the mean squared error (MSE).

8. Model Evaluation:

The trained model is evaluated on the test set. Metrics such as Mean Squared Error (MSE), Absolute Error, and R2 Score are calculated to assess the model's performance in predicting median house values.

For my model the following values are obtained:

```
Epoch 0, MSE: 0.1993
Epoch 100, MSE: 0.0333
Epoch 200, MSE: 0.0267
Epoch 300, MSE: 0.0237
Epoch 400, MSE: 0.0219
Epoch 500, MSE: 0.0205
Epoch 600, MSE: 0.0193
Epoch 700, MSE: 0.0183
Epoch 800, MSE: 0.0175
Epoch 900, MSE: 0.0168
Final Epoch 1000, MSE: 0.0162
Mean Squared Error on Test Set: 0.0148
Absolute Error on Test Set: 0.0807
R2 Score on Test Set: 0.5903
```