PRML Assignment #2

Parth Darshan (B22CS040

Before Starting, Necessary type of libraries have been imported.

Q.1

- **Loading the Dataset :** Load Titanic dataset, suppress warnings, display first rows, and show column names using pandas.

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

- **Check for Missing Values** : Print percentage of missing values for each column in Titanic dataset using pandas.

Here It can be observed that Age has approximately 20 % of the data empty, so we handle these values by filling the data with mean age.

Cabin column is majorly empty so, we can swiftly drop the column.

In the Embarked Column, only 0.225% rows are empty, we can swiftly drop those rows.

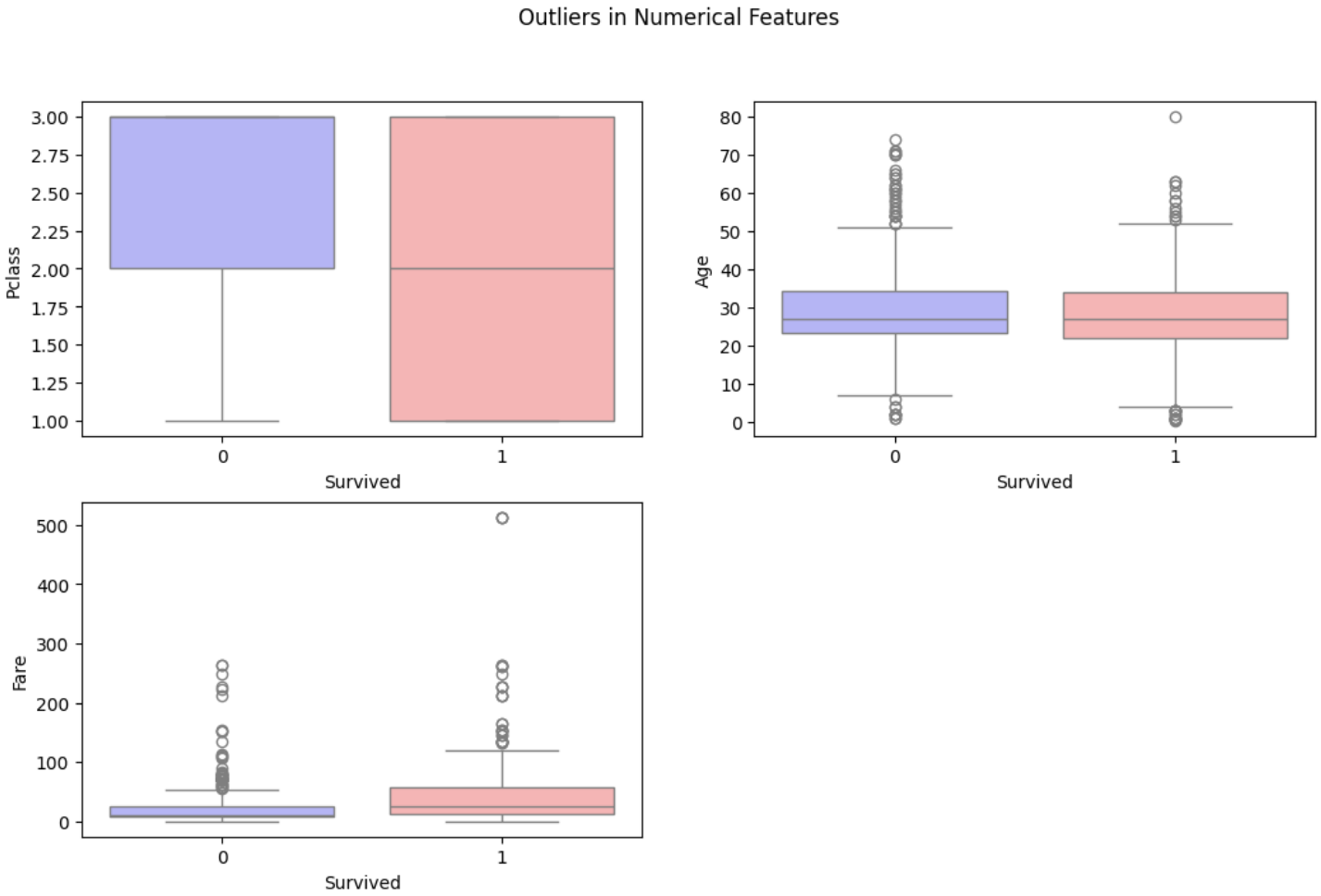
In this dataset, PClass is ORDINAL Dataset
Age, Name, PassengerID, Ticket, Fare are Continuous
Rest Columns are Nominal Dataset

```
Percentage of missing values:
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            19.865320
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.000000
Cabin          77.104377
Embarked        0.224467
dtype: float64
```

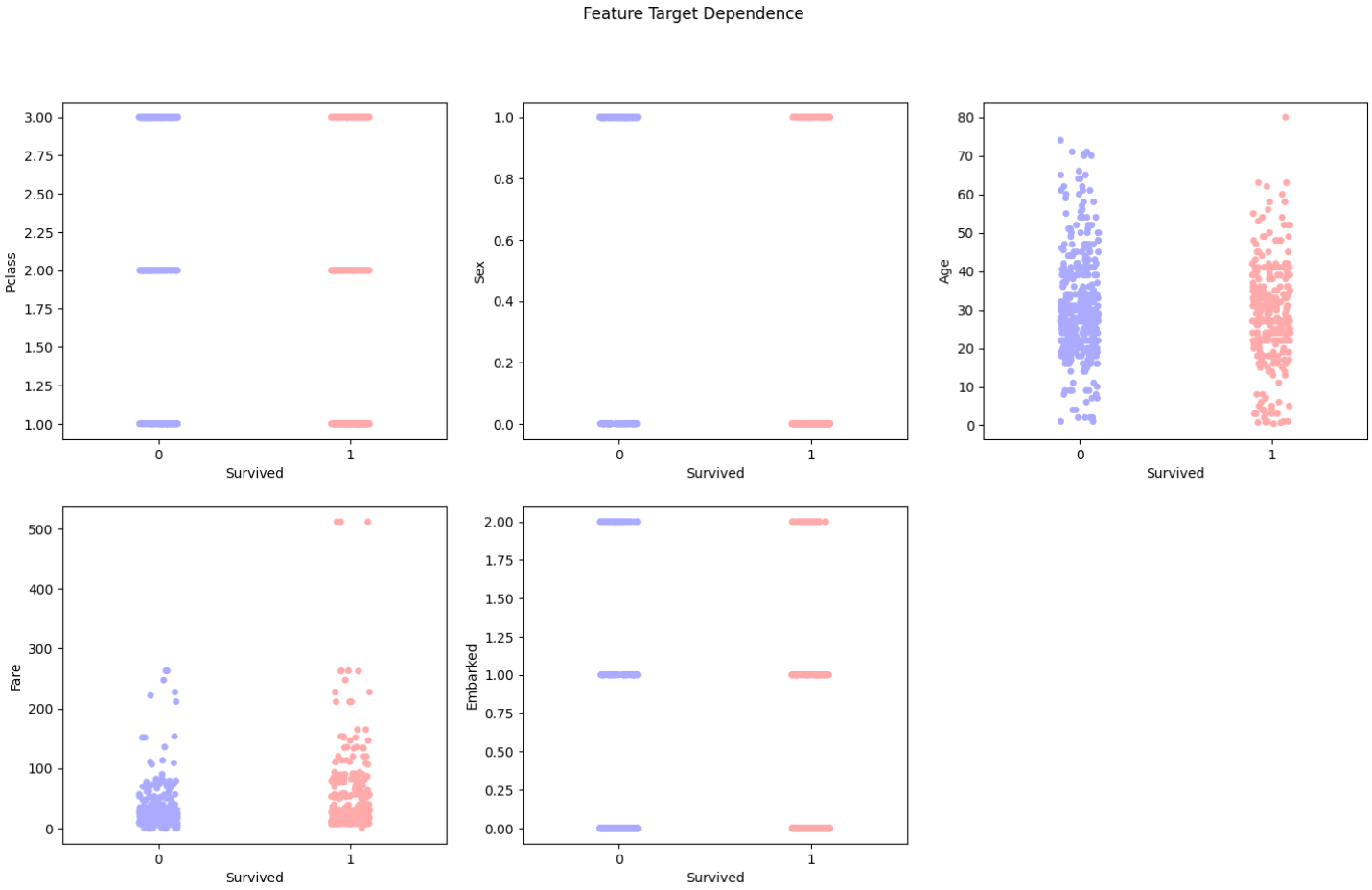
- Verify if all missing data has been handled.

```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

- **Dropping Unnecessary Columns :** Columns such as “PassengedID”, “Name”, “Ticket” were drop as they were unnecessary.
- **Visualize outliers in numerical features :** Create a 2x2 subplot, visualize outliers using boxplots. Top-left: 'Pclass' vs. 'Survived', top-right: 'Age' vs. 'Survived', bottom-left: 'Fare' vs. 'Survived'. The last subplot is removed for simplicity.



- **Conversion to Categorical Data :** Convert categorical features 'Sex' and 'Embarked' to numerical values: 'female' to 0, 'male' to 1; 'S' to 0, 'C' to 1, 'Q' to 2.
- **Display feature-target dependence :** The plot visualizes the relationship between survival and various features. It includes subplots for passenger class, gender, age, fare, and embarkation port, offering insights into how these factors may be associated with survival outcomes on the Titanic.



- **Splitting the Data :** The code splits the dataset into training, validation, and test sets using `train_test_split`. It prints the shapes of the resulting sets, providing insights into the distribution of data for modeling.

```
Shape of X_train: (622, 7)
Shape of y_train: (622,)
Shape of X_test: (89, 7)
Shape of y_test: (89,)
```

- **Implementing the Decision Tree :** The provided code defines a basic Decision Tree classifier from scratch and applies it to the Titanic dataset for binary classification of survival. The DecisionTree class is implemented with methods for fitting the model, predicting, and building the tree recursively. The decision tree is grown based on the Gini impurity criterion, and the tree depth and minimum samples per leaf are configurable parameters.

The dataset is loaded and features 'Pclass', 'Sex', and 'Age' are selected for training. An instance of the DecisionTree class is created, with a specified maximum depth of 3 and a minimum samples per leaf set to 5. The fit method is then called to train the model on the selected features and the target variable 'Survived'.

Subsequently, predictions are made on the training set using the predict method of the DecisionTree class. The predicted values are printed for further examination or evaluation.

It's important to note that this Decision Tree implementation is simplified and lacks some optimization and functionalities present in more sophisticated implementations, such as those

provided by machine learning libraries like scikit-learn. Nevertheless, the code offers a fundamental understanding of the decision tree algorithm and its application for binary classification.

- **Evaluating the Result:** The provided code defines three functions for evaluating a classification model: `confusion_matrix`, `accuracy_score`, and `classification_report`. The `confusion_matrix` function generates a matrix showing the counts of true positive, true negative, false positive, and false negative predictions. The `accuracy_score` function calculates the accuracy of the model by dividing the number of correct predictions by the total number of predictions. The `classification_report` function computes precision, recall, F1-score, and support for each class in a multiclass classification problem based on the confusion matrix. Finally, the evaluation metrics are calculated and printed for the predicted values compared to the actual values on the test set.

```
Confusion Matrix:
{0: {0: 37, 1: 16}, 1: {0: 27, 1: 9}}
Accuracy: 51.68539325842697
Classification Report:
{'precision': {0: 0.578125, 1: 0.36}, 'recall': {0: 0.6981132075471698, 1: 0.25}, 'f1-score': {0: 0.6324786324786325, 1: 0.2950819672131475},
```

Q.2

This documentation provides information about a Python code snippet that demonstrates simple linear regression using a dataset related to TV marketing. The code utilizes the pandas library to read a CSV file from a specified URL, and then it prints the shape and the first few rows of the resulting Data Frame df2.

- **Importing the necessary libraries:** For this question, we needed to import NumPy, Pandas and Matplotlib. NumPy is used for calculations involved in solving the question, Pandas is used for data manipulation and Matplotlib is used for plotting the graphs required.
- **Importing the data:** For importing, I used URL to import the data file. After importing the file, we check for its shape and analyze the dataset by viewing few of the entries in the data frame.

```
(200, 2)
```

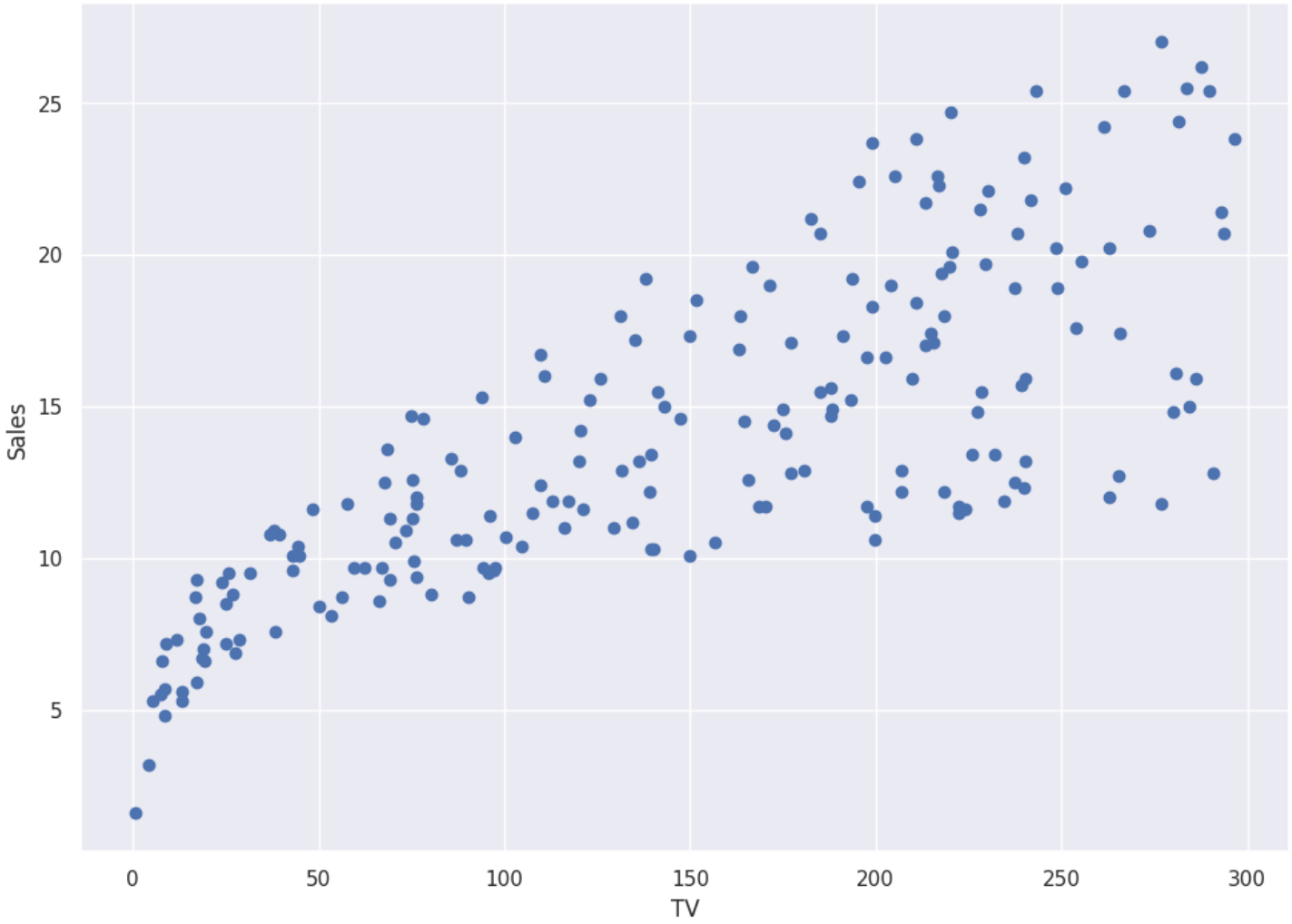
	TV	Sales
0	230.1	22.1
1	44.5	10.4
2	17.2	9.3
3	151.5	18.5
4	180.8	12.9

The Data frame is a table containing 200 rows and 2 columns as shown by “`df2.shape`”.

- **Calculating the Mean and Standard Deviation:** The next snippet of code finds the mean and the standard deviation of both the columns. To find the mean and standard deviation, I used NumPy to calculate each quantity.

```
The mean of TV is :  147.0425
The mean of Sales is :  14.0225
The SD of TV is :  85.63933175679269
The SD of Sales is :  5.204396578855228
```

- **Plotting the Scatter Plot:** Plotted the scatter plot of the dataset using Matplotlib



- **Getting to know the dataset better:** Found out more information about the dataset

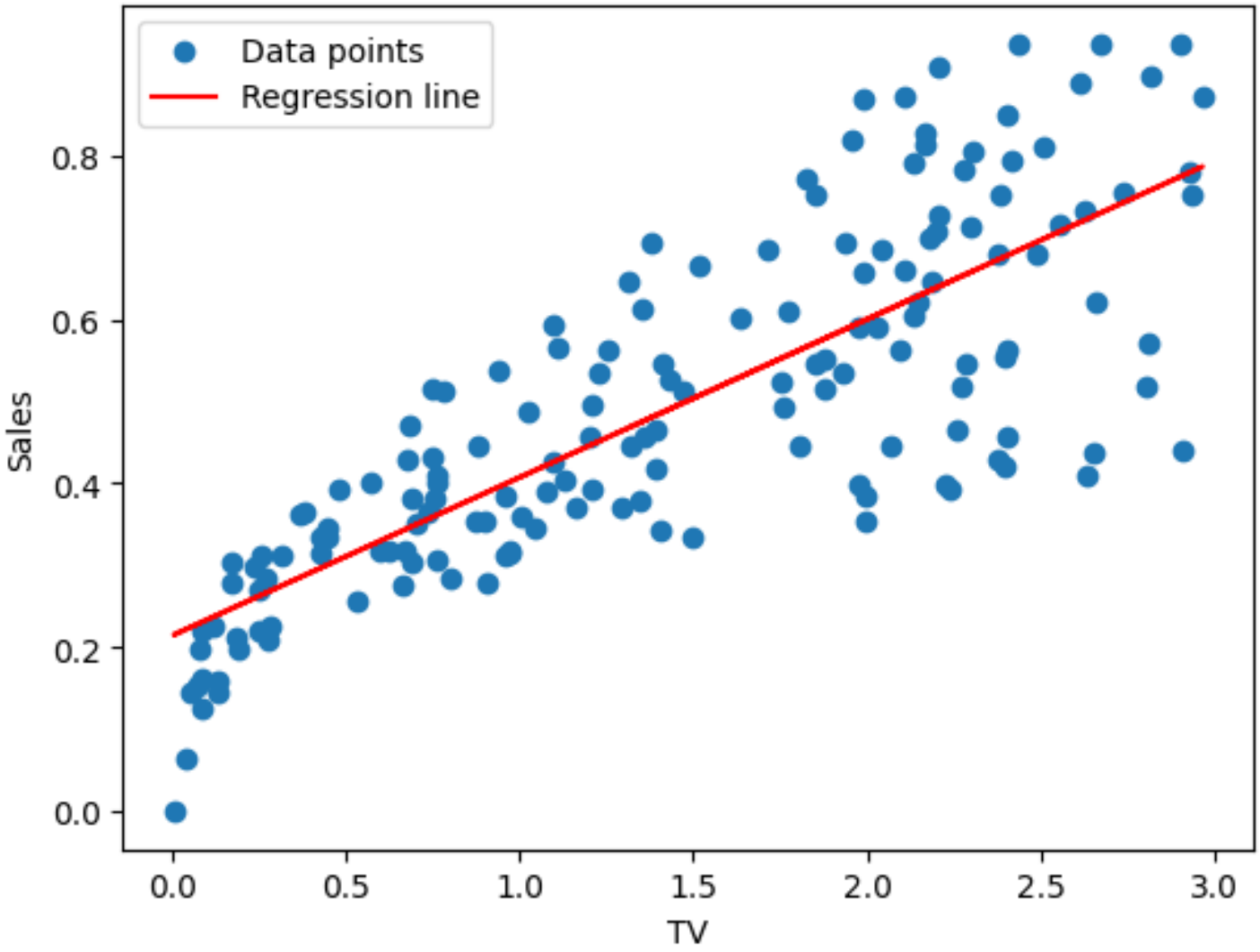
As per above data, there is no null value present in the dataset . Thus, no need to handle null values (NaN).

```
<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
```

- **Normalizing the data:** For Normalization, I applied Min-Max Normalization on the Y axis and Linear scaling (dividing by 100) on the X axis. The Y axis represents sales and X axis represents TV.
- **Splitting the data :** Normal Splitting by introducing a 'train_length' variable to divide the data. First 160 entries are in the training dataset while the rest have been set to the testing dataset.
- **Linear Regression :** I applied Linear Regression with the help of gradient descent. I have also used minimum squared cost function to implement Linear Regression. The function requires two arguments, the variable on the X axes and the Y axes respectively. Slope and Intercept are returned.

```
Slope :  0.19342323992749325
Y intercept :  0.21393486127119452
```

- **Plotting the Data:** Self explanatory



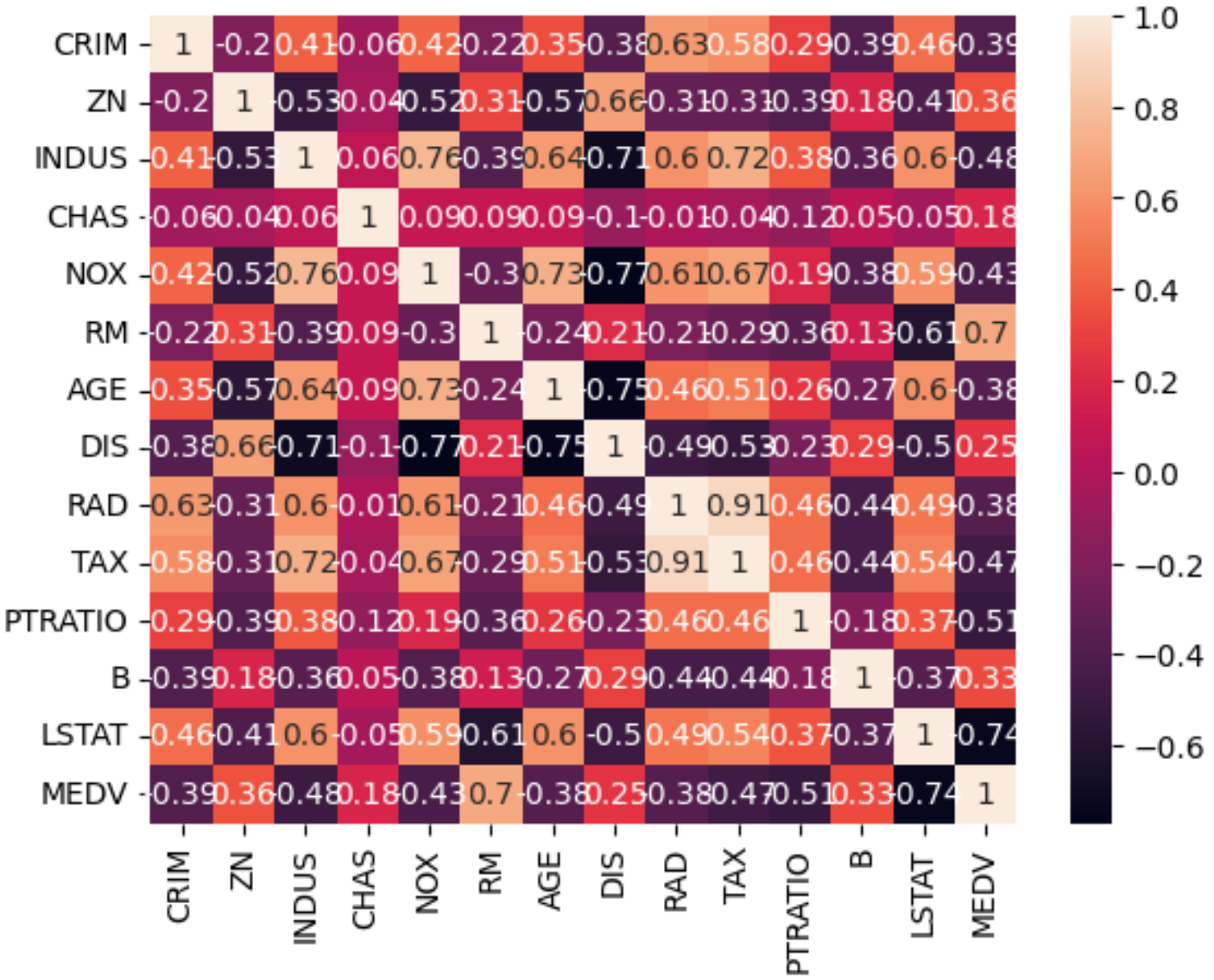
- **Calculating Error :** The MSE and Absolute error are:

```
Mean Squared Error: 0.02190715365812915
Absolute Error: 0.12121283587490532
```

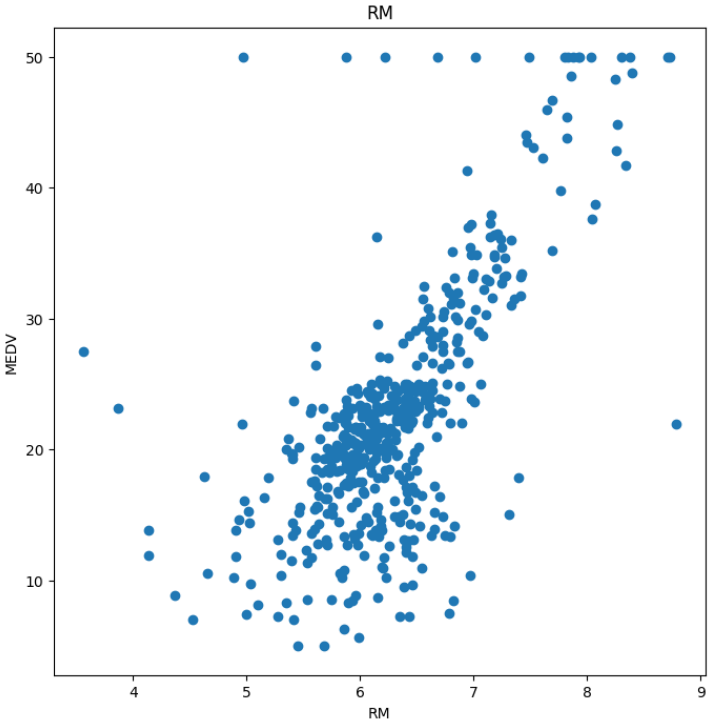
Q.3

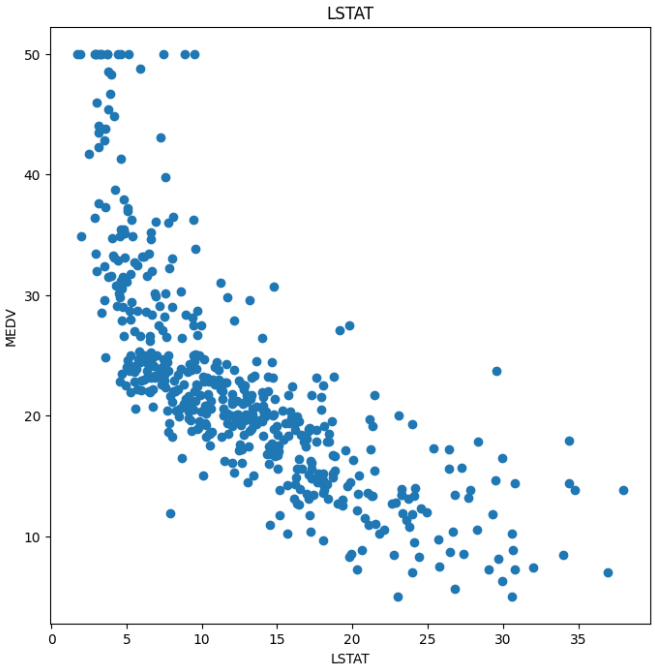
- **Loading the data :** The code begins by specifying the correct URL for the Boston Housing dataset in CSV format and defining column names. The Pandas library is then used to read the dataset into a DataFrame, addressing missing values by replacing them with the mean values of their respective columns. This ensures a complete and cleaned dataset for further analysis. The paragraph succinctly encapsulates the essential steps of loading and preprocessing the data, laying the foundation for subsequent exploratory data analysis and model training.
- **Exploratory Data Analysis :** Following data loading and preprocessing, the code conducts Exploratory Data Analysis (EDA) on the Boston Housing dataset. It showcases the first few rows of the DataFrame and provides visual insights into the distribution of the target variable, 'MEDV', through a histogram. Additionally, a correlation matrix heatmap is generated to highlight relationships between different features. These visualizations aid in understanding the dataset's

structure, identifying potential patterns, and informing subsequent modeling decisions. The EDA section sets the stage for informed feature selection and model interpretation, contributing to a comprehensive understanding of the dataset's characteristics.



- **Data normalization**: Achieved by scaling features to a consistent range, enhances model performance by preventing any single variable from dominating. It ensures fair contributions and efficient convergence during model training.





- **Linear Regression Implementation** : The code implements a simple Linear Regression model for predicting house prices in the Boston Housing dataset. The `LinearRegression` class is designed with parameters for learning rate and epochs, and it includes methods for fitting the model to training data and making predictions. The model employs gradient descent to iteratively update weights and bias, minimizing the Mean Squared Error (MSE). During training, MSE is monitored and recorded for each epoch. The model's architecture allows flexibility for hyperparameter tuning and observation of the learning process. This linear regression implementation serves as a fundamental predictive tool, laying the groundwork for further enhancements and evaluations. Its simplicity makes it suitable for educational purposes, providing a hands-on understanding of the core concepts underlying linear regression in machine learning.
- **Model Evaluation and Visualization :** The code evaluates the trained Linear Regression model on a test set, calculating the Mean Squared Error (MSE) to gauge predictive accuracy. This metric quantifies the average squared difference between actual and predicted values. The resulting

MSE on the test set is then displayed. To enhance comprehension, a visual representation is crafted, juxtaposing actual house prices against predicted values for a specific feature, 'RM' (Number of Rooms). This scatter plot aids in assessing the model's predictive performance, offering a clear comparison between observed and predicted outcomes on the test data and providing valuable insights into the model's effectiveness.