PRML LAB ASSIGNMENT REPORT

ANUSHKA KUNJ DADHICH
B22CS097

QUESTION 1

Task 1 -

Preprocessing and data visualization -

Replaced the empty cells by various methods -

1. Dropped column with heavy empty cells
2. For very fewer empty cells, dropped the rows, without heavy loss of data
3. For mediocre empty cells (age), fill it by the average values categorized into various subsets.

After preprocessing percentage of data with null values-

```
PassengerId    0.0
Survived       0.0
Pclass         0.0
Name           0.0
Sex            0.0
Age            0.0
SibSp          0.0
Parch          0.0
Ticket         0.0
Fare           0.0
Embarked       0.0
dtype: float64
```

Encoding like - Survived class - converted into binary 0s for not survived and 1s for survived
Converted sex into 0s for males and 1s for females
Embarked into 0s, 1s and 2s for specific letters
Pclass - already discrete values

Output -

```
Discrete values in Pclass: [3 1 2]
```

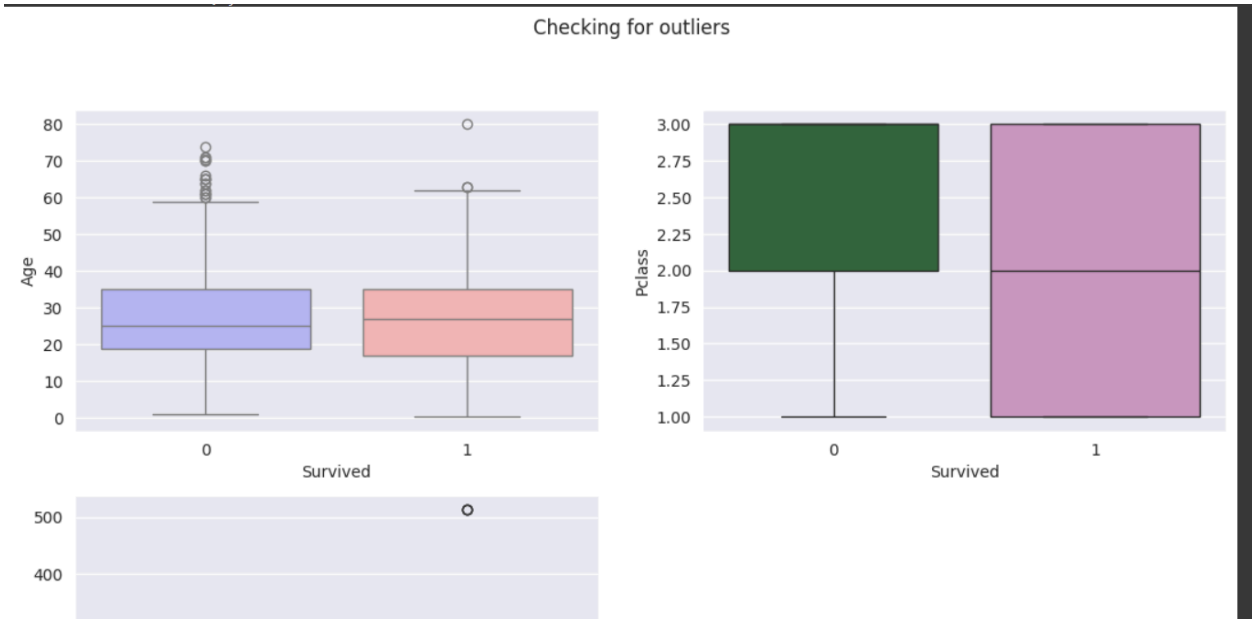
```
Sex before encoding:
0      male
1    female
2    female
3    female
4      male
Name: Sex, dtype: object
Sex after encoding
0    0
1    1
2    1
3    1
4    0
Name: Sex, dtype: int64
```

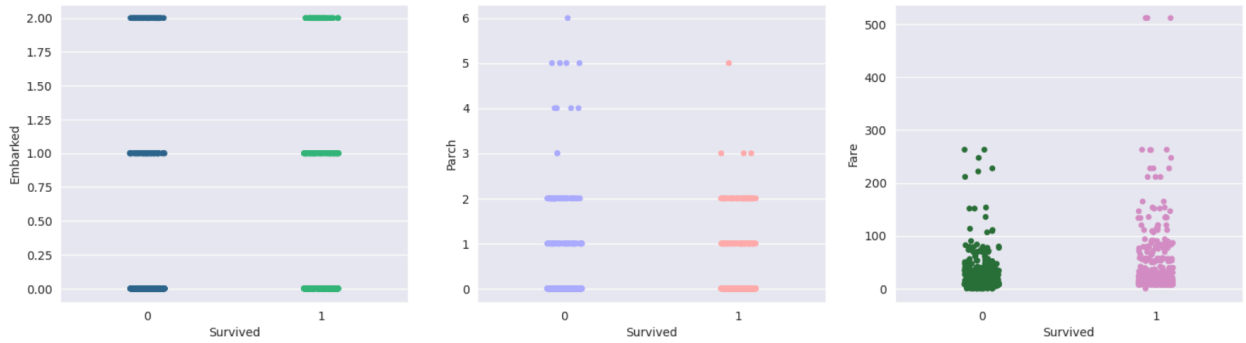
```
Column 'Embarked' before encoding:
0    S
1    C
2    S
3    S
4    S
Name: Embarked, dtype: object
Column 'Embarked' after encoding:
0    0
1    1
2    0
3    0
4    0
Name: Embarked, dtype: int64
```

FEATURES	**TYPES**
Pclass	Ordinal
Sex	Categorical
Age	Nominal

SibS	Categorical
PArch	Categorical
Fare	Continuous
Embarked	Nominal

Using seaborn library, drew the various plots of survived class with other features to show the variations and check for outliers - No outliers found





Data splitting into test, train and validate using train_test_split() twice, once for test and (train+validate) and then splitting into train and validate.
Output-

```
Features:
    Pclass  Sex   Age  SibSp  Parch     Fare  Embarked
0       3    0  22.0      1      0   7.2500         0
1       1    1  38.0      1      0  71.2833         1
2       3    1  26.0      0      0   7.9250         0
3       1    1  35.0      1      0  53.1000         0
4       3    0  35.0      0      0   8.0500         0
Target:
 0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

```
Shape of X_train: (624, 7)
Shape of y_train: (624,)
Shape of X_test: (89, 7)
Shape of y_test: (89,)
Shape of X_val: (89, 7)
Shape of y_val: (89,)
Training set percentage: 70.19%
Validation set percentage: 19.80%
Test set percentage: 10.01%
```

Task 2:

Created functions -

1. To calculate entropy - using: entropy = -np.sum(probabilities * np.log2(probabilities))
2. To find information gain - using the feature and the threshold as input, it would calculate information gain using the above function
3. To find threshold - calculating information gain for every unique value in the feature and return value at which maximum entropy gain is there

Output -
Just functions created

Task 3:

Created the req function using the above functions in task 2, returning 0s below the threshold and 1s above the threshold.
Outputs-

```
Optimal Threshold for Age: 15.0
```

```
# # example
conTocat(X,'Age')
```

```
[1,
 1,
 1,
 1,
 1,
 1,
 1,
 0,
 1,
 0,
 0,
 1,
 1,
 1,
 0,
 1,
 0,
 1,
 1,
 1,
 1,
 1,
```

Task 4:

1. get_best_split(X,y,available_features) - return best split using the function to get optimal threshold
2. get_majority_class(y) - returns the class with maximum unique labels used at the leaf node.
3. build_tree(X, y, depth=0, max_depth=None) - Repeat the tree creating steps for the every newly-created split parts, using the max depth as stopping criteria.

Output -
Created tree

```
{'feature': 'Sex', 'threshold': 0, 'left': {'feature': 'Age', 'threshold': 9.0, 'left': {'feature': 'SibSp', 'threshold': 2, 'left': {'class': 1}, 'right': {'class': 0}}, 'right': {'fe
```

Task 5:

infer(sample,tree) - iterates through the decision tree until the root node and returns the root node class for the passed in sample using the threshold values.
Output -
Just function creation which is used in next task

```
def infer(sample,tree):
 if 'feature' in tree:
   feature_value = sample.get(tree['feature'], 0)
   if feature_value <= int(tree['threshold']):
       # into the left subtree
       return infer(sample, tree['left'])
   else:
       # into the right subtree
       return infer(sample, tree['right'])
 else:
   # If the current node doesn't have a feature,
   return tree['class']
```

Task 6:

1. calculate_accuracy(true_labels, predicted_labels) - calculate and pass predicted label using infer, and then this maps and calculates if both the values match or not and then divided it with the total samples
2. calculate_classwise_accuracy(true_labels,predicted_labels) - first finds the unique labels in the true labels, and then repeats the same calculations for all labels

Output -

```
Overall Accuracy on Training Set: 0.8253205128205128
Classwise Accuracy on Training Set: {0: 0.884318766066838, 1: 0.7276595744680852}
Overall Accuracy on Test Set: 0.8314606741573034
Classwise Accuracy on Test Set: {0: 0.9038461538461539, 1: 0.7297297297297297}
```

Task 7:

calculate_confusion_matrix(true_labels,predicted_labels) - finds the count of labels which match each other - 0s return the tn, 1s return the tp and then those which do not match with each other and true label as 1 are fp and those with 0 are fn. And then return the np matrix
Output -

```
[[47 10]
 [ 5 27]]
```

Task 8:

Created functions for these values:

1. get(true_labels,predicted_labels) - returns the values of tp, tn, fn and fp using the same logic in the confusion matrix function
2. calculate_precision(true_labels,predicted_labels) - using the data from get() function it returns the value of ((tp)/(tp+fp))
3. calculate_recall(true_labels,predicted_labels) - using the data from get() function it returns the value of ((tp)/(tp+fn))
4. calculate_f1(true_labels,predicted_labels) - using the data from above 2 functions it returns the value of HM i.e. ((2*p*r)/(p+r))

Output -

```
Precision of test data =  0.7297297297297297
Recall of test data =  0.84375
F1 score of test data =  0.7826086956521738
```

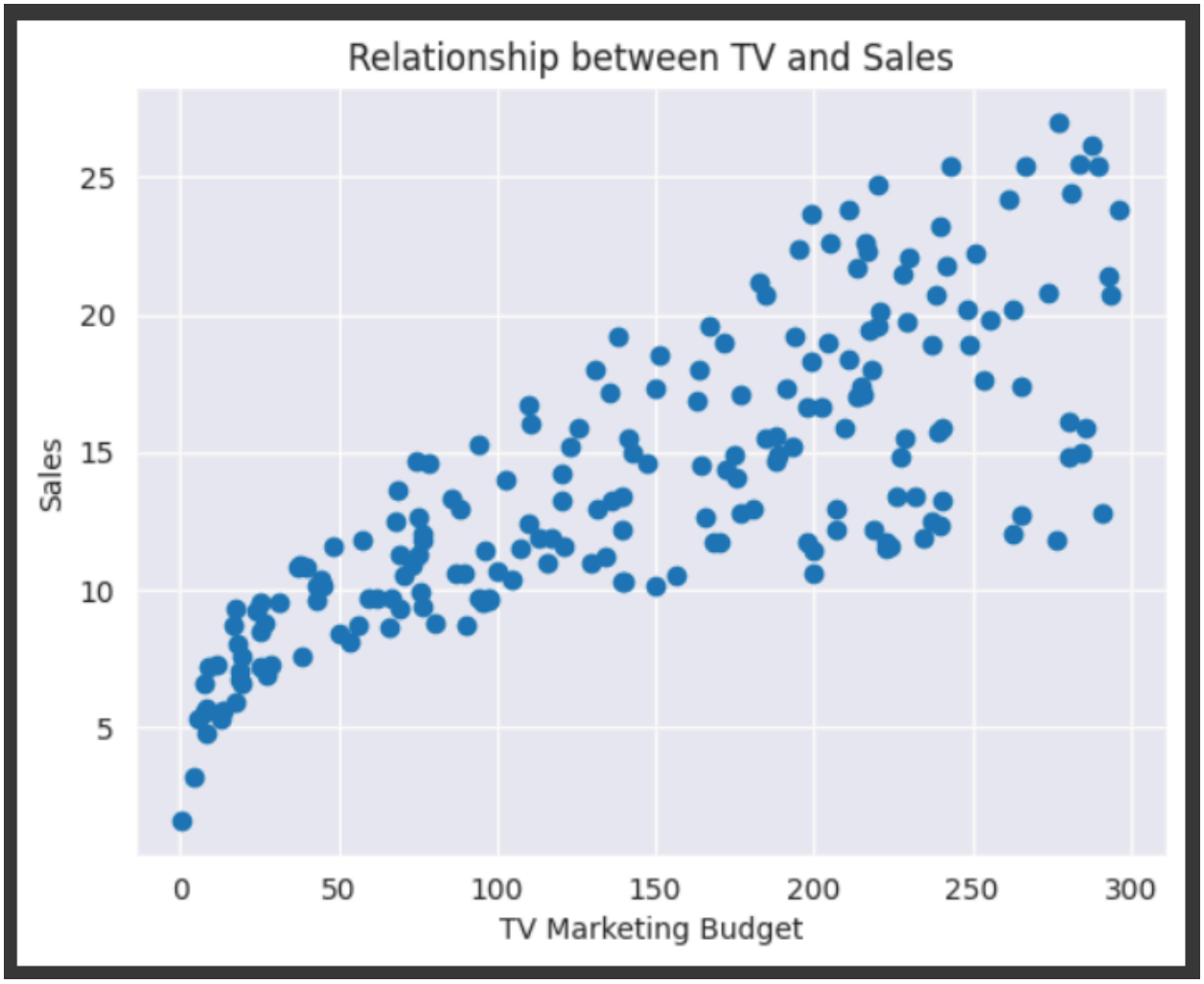
QUESTION 2

Task 1 -

1. Using read_csv for storing data in df using pandas and then df.head() for displaying the first 5 rows.
2. Using matplotlib 'plt.scatter(df['TV'],df['Sales'])' to display the scatter plot of the data
   Inference - data is scattered in a cone type of shape, and as the TV budget increases the sales also increases.
3. Using .mean() and .std() respectively for calculating the means and standard deviations of the columns

Output -

	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




```
Mean of the TV Marketing Budget column =  147.0425
Mean of the Sales column =  14.0225
Standard Deviation of the TV Marketing Budget column =  85.85423631490808
Standard Deviation of the Sales column =  5.217456565710478
```

Task 2 -

1. Using .isna() to check for missing values in the data - Data has no missing values
2. Normalizing the contents using Z-score normalization by creating a function for the same
3. Using train_test_split(X1, y1, test_size=0.2) to split the given data

Output -

```
Percentage of missing values:
TV       0.0
Sales    0.0
dtype: float64
```

No missing values in the data

```
Before normalization -
      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
After normalization -
         TV     Sales
0  0.967425  1.548168
1 -1.194379 -0.694304
2 -1.512360 -0.905135
3  0.051919  0.858177
4  0.393196 -0.215143
```

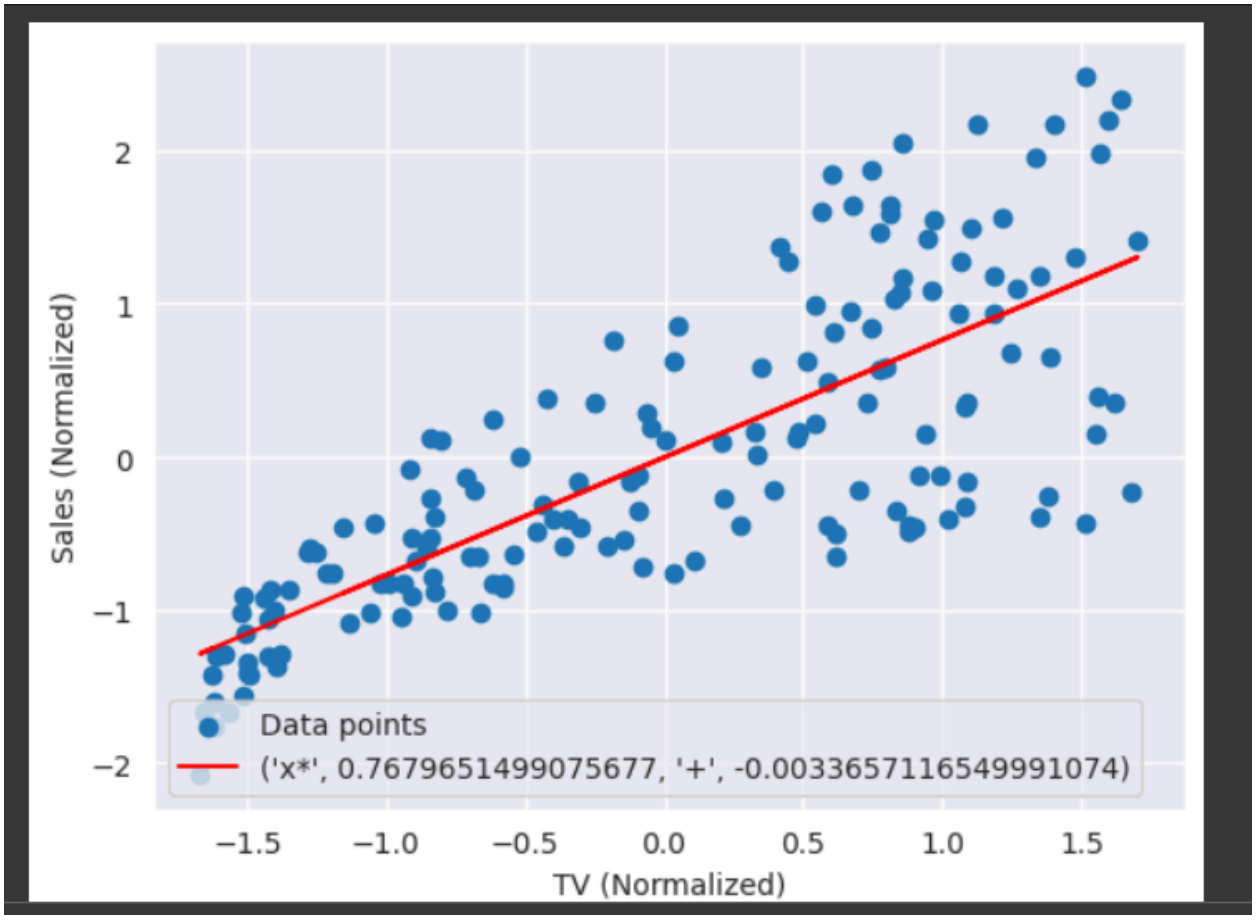
```
Shape of X_train: (160,)
Shape of y_train: (160,)
Shape of X_test: (40,)
Shape of y_test: (40,)
Training set percentage: 80.00%
Test set percentage: 20.00%
```

Task 3 -

1. hypo(x,w1,w0) - returns the hypothesis function (y = x*w1+w0)
2. cost_fn(X,y,w1,w0) - calculates the MSE as cost function
3. gradient_descent(X, y, w1, w0, learning_rate, epochs) - applies the gradient descent algorithm on the training dataset. It iteratively updates the parameters w1 and w0 based on the gradients of the cost function with respect to these

parameters. The learning rate determines the size of the steps taken during each iteration, and the process is repeated for the specified number of epochs.

4. Plotted the line on the scatter plot using matplotlib.



Task 4 -

1. mean_squared_error(y_true, y_pred) - calculates the MSE of the predicted and true data
2. absolute_error(y_true, y_pred) - calculates the absolute error using the predicted and true data

```
Mean Squared Error on Test Set: 0.3267283396635702
Absolute Error on Test Set: 0.4592108223054206
```

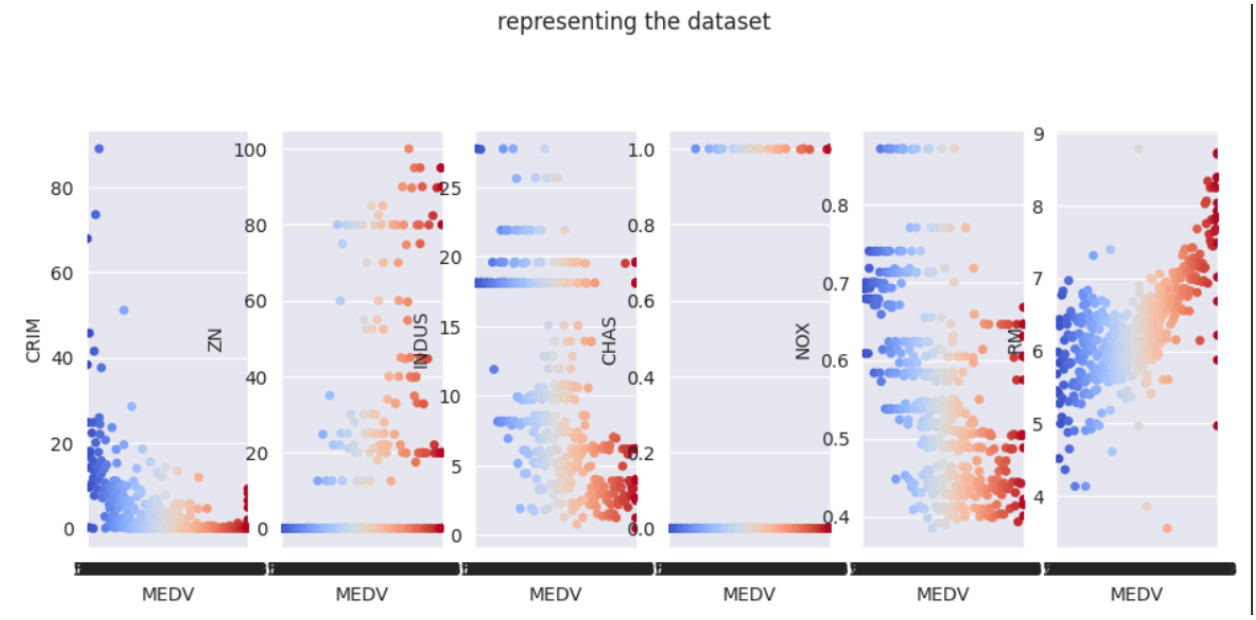
QUESTION 3

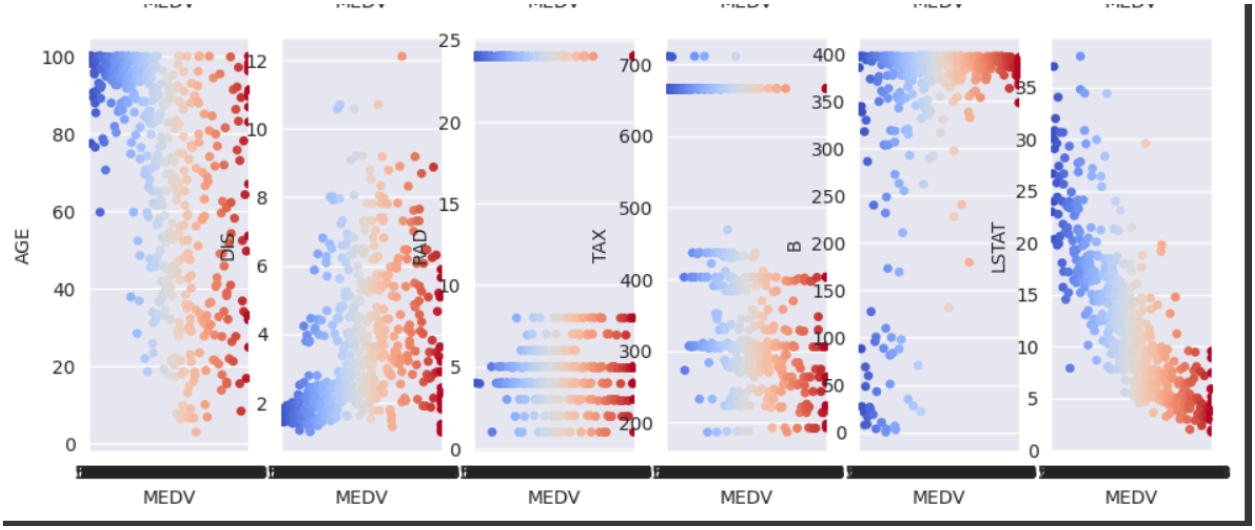
Task 1 -

1. Using read_csv for storing data in df using pandas and then df.head() for displaying the first 5 rows.
2. Using matplotlib 'plt.scatter(df['TV'],df['Sales'])' to display the scatter plot of the data
   Inference - data is scattered in both increasing and decreasing orders for various features.
3. Using .mean() and .std() respectively for calculating the means and standard deviations of the columns

Outputs -

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	NaN	36.2






```
              CRIM            ZN         INDUS          CHAS           NOX  \
mean  2.924044e-17  2.193033e-17  1.315820e-16 -2.924044e-17 -2.527622e-16
std   1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00

                RM           AGE           DIS  RAD           TAX  \
mean -9.478584e-17 -3.655055e-17 -1.404235e-16  0.0  5.616939e-17
std   1.000000e+00  1.000000e+00  1.000000e+00  1.0  1.000000e+00

             PTRATIO             B         LSTAT          MEDV
mean -4.212704e-16 -7.021173e-16 -1.169618e-16 -5.476515e-16
std   1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
```

Task 2 -

1. Using .isna() to check for missing values in the data - Replaced the empty cells by the mean value of that column
2. Normalizing the contents using Z-score normalization by creating a function for the same
3. Using train_test_split(X1, y1, test_size=0.2) to split the given data

Output -

Missing values before replacement -

```
Percentage of missing values:
CRIM       3.952569
ZN         3.952569
INDUS      3.952569
CHAS       3.952569
NOX        0.000000
RM         0.000000
AGE        3.952569
DIS        0.000000
RAD        0.000000
TAX        0.000000
PTRATIO    0.000000
B          0.000000
LSTAT      3.952569
MEDV       0.000000
dtype: float64
```

Missing values after replacement -

```
Percentage of missing values:
CRIM       0.0
ZN         0.0
INDUS      0.0
CHAS       0.0
NOX        0.0
RM         0.0
AGE        0.0
DIS        0.0
RAD        0.0
TAX        0.0
PTRATIO    0.0
B          0.0
LSTAT      0.0
MEDV       0.0
dtype: float64
```

```
Before normalization -
      INDUS        RM   PTRATIO         LSTAT      MEDV
0 -1.283517  0.413263 -1.457558 -1.080991e+00  0.159528
1 -0.587193  0.194082 -0.302794 -4.996502e-01 -0.101424
2 -0.587193  1.281446 -0.302794 -1.213749e+00  1.322937
3 -1.302535  1.015298  0.112920 -1.366072e+00  1.181589
4 -1.302535  1.227362  0.112920 -1.169618e-16  1.486032
After normalization -
      INDUS        RM   PTRATIO         LSTAT      MEDV
0 -1.309714  0.413263 -1.457558 -1.103054e+00  0.159528
1 -0.599178  0.194082 -0.302794 -5.098482e-01 -0.101424
2 -0.599178  1.281446 -0.302794 -1.238522e+00  1.322937
3 -1.329120  1.015298  0.112920 -1.393954e+00  1.181589
4 -1.329120  1.227362  0.112920  9.611603e-18  1.486032
```

```
Shape of X_train: (404, 4)
Shape of y_train: (404,)
Shape of X_test: (102, 4)
Shape of y_test: (102,)
Training set percentage: 79.84%
Test set percentage: 20.16%
```

Task 3 -

1. hypo(x,weights,w0) - returns the hypothesis function (y = x*(weights)+w0)
2. cost_fn(X,y,weights,w0) - calculates the MSE as cost function
3. gradient_descent(X, y, weights, w0, learning_rate, epochs) - applies the gradient descent algorithm on the training dataset. It iteratively updates the parameters weights and w0 based on the gradients of the cost function with respect to the parameters. The learning rate is the size of the steps taken during each iteration, and the process is repeated for the specified number of epochs.
4. Plotting the data points on the scatter plot (of single feature) as actual and predicted values, the overlap shows the accuracy of the model.

Outputs -

Correlation -

```
dataset.corr()['MEDV']
```

```
CRIM      -0.379695
ZN         0.365943
INDUS     -0.478657
CHAS       0.179882
NOX       -0.427321
RM         0.695360
AGE       -0.380223
DIS        0.249929
RAD       -0.381626
TAX       -0.468536
PTRATIO   -0.507787
B          0.333461
LSTAT     -0.721975
MEDV       1.000000
Name: MEDV, dtype: float64
```

Kept dataset -

	INDUS	RM	PTRATIO	LSTAT	MEDV
0	-1.283517	0.413263	-1.457558	-1.080991e+00	0.159528
1	-0.587193	0.194082	-0.302794	-4.996502e-01	-0.101424
2	-0.587193	1.281446	-0.302794	-1.213749e+00	1.322937
3	-1.302535	1.015298	0.112920	-1.366072e+00	1.181589
4	-1.302535	1.227362	0.112920	-1.169618e-16	1.486032
...	...	...	...	...	...
501	0.123760	0.438881	1.175303	-1.169618e-16	-0.014440
502	0.123760	-0.234316	1.175303	-5.080349e-01	-0.210154
503	0.123760	0.983986	1.175303	-9.887591e-01	0.148655
504	0.123760	0.724955	1.175303	-8.713729e-01	-0.057932
505	0.123760	-0.362408	1.175303	-6.757294e-01	-1.156104

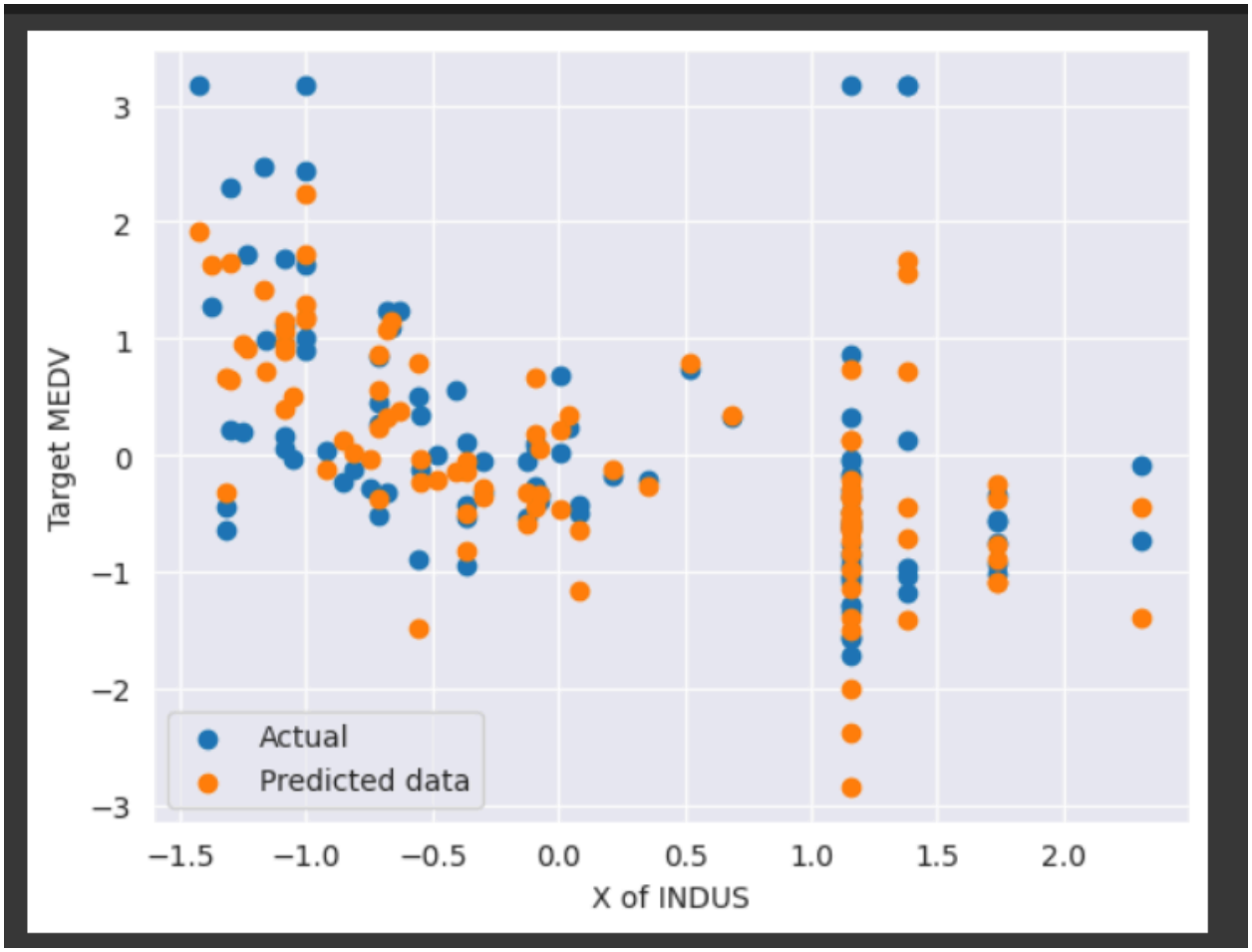
506 rows × 5 columns

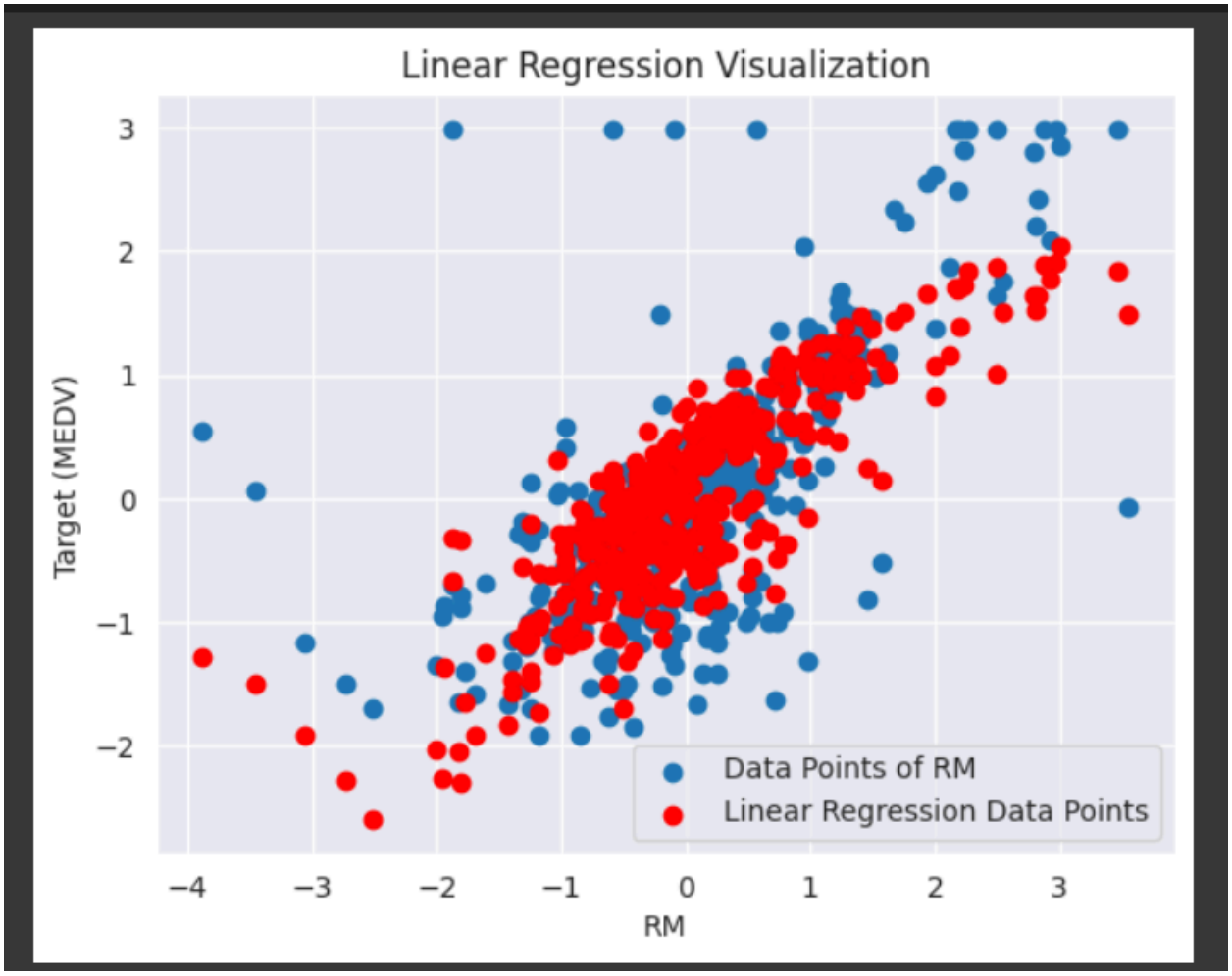
```
learning_rate = 0.01
epochs = 1000

weights, w0 = gradient_descent(X3_train,y3_train,learning_rate,epochs)
print(weights,w0)
```

```
[-0.0439341   0.35755881 -0.20438872 -0.42311455] -0.014482721984988869
```

Graph -


Target MEDV
X of INDUS
Actual
Predicted data
3
2
1
0
−1
−2
−3
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0



Task 4 -

1. mean_squared_error(y_true, y_pred) - calculates the MSE of the predicted and true data
2. absolute_error(y_true, y_pred) - calculates the absolute error using the predicted and true data

Outputs -

```
Mean Squared Error on Test Set: 0.35145531154177 39
Absolute Error on Test Set: 0.4256213647891338
```