REPORT
ASSIGNMENT 2

LAVANGI PARIHAR
B22EE044

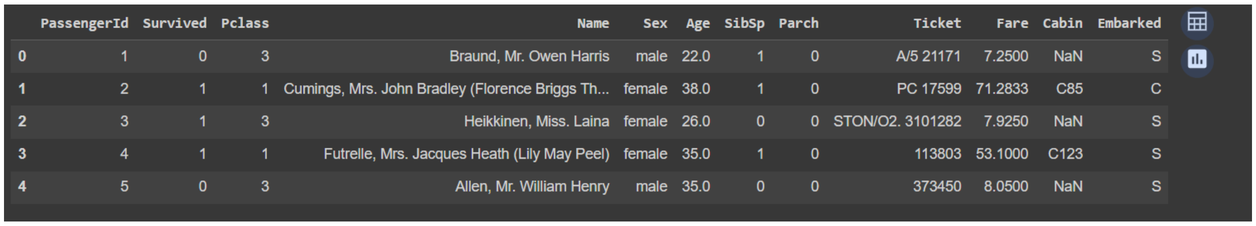
<u>QUESTION 1</u>
Implementing a Decision Tree

Task 1 Data Pre Processing:

- <u>Initial Data Visualization</u>

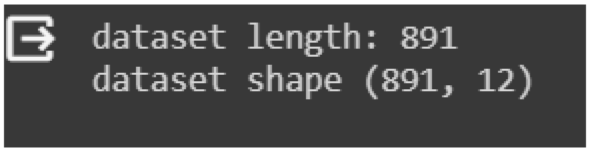
  1. Loading Data:
     - Titanic dataset loaded from "titanic.csv" using Pandas.

  2. Initial Overview:
     - Displayed the first five rows for a quick glance at the data.



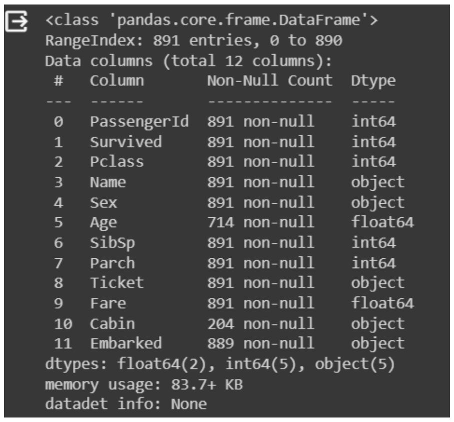
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 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

  3. Dataset Size:
     - Dataset contains [number of rows] entries.
  4. Data Shape:
     - Structure: [number of rows, number of columns].



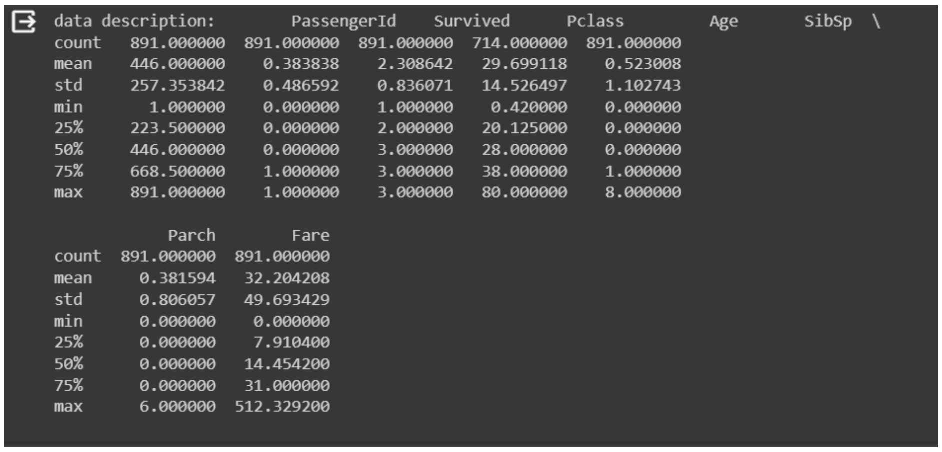
```
dataset length: 891
dataset shape (891, 12)
```

  5. Data Info:
     - Data types, non-null counts, and memory usage overview provided.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
datadet info: None
```

6. Data Description:
- Statistical summary (mean, std, min, max) for numerical columns.



```
data description:        PassengerId    Survived      Pclass         Age       SibSp  \
count    891.000000  891.000000  891.000000  714.000000  891.000000
mean     446.000000    0.383838    2.308642   29.699118    0.523008
std      257.353842    0.486592    0.836071   14.526497    1.102743
min        1.000000    0.000000    1.000000    0.420000    0.000000
25%      223.500000    0.000000    2.000000   20.125000    0.000000
50%      446.000000    0.000000    3.000000   28.000000    0.000000
75%      668.500000    1.000000    3.000000   38.000000    1.000000
max      891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

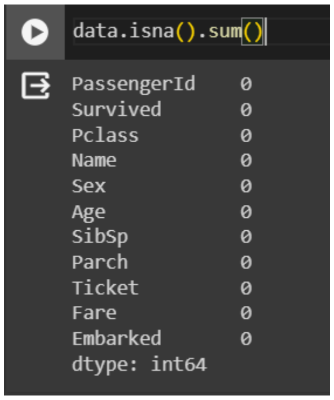
- Handling Missing values

  1. Imputation of Missing Values in 'Age':
     - The script employs a two-step process to handle missing values in the 'Age' column.
     - First, it calculates the mean age for different groups defined by the combination of 'Sex' and 'Pclass'.
     - Second, it iterates through the dataset and fills missing 'Age' values based on specific conditions for 'Sex' and 'Pclass'. This conditional imputation aims to provide more accurate estimations than a generic approach.
     -This is done because the missing value percentage in age column is neither too high nor too low.

  2. Removal of 'Cabin' Column:
     - The decision to drop the 'Cabin' column is based on its high percentage of missing values. By removing this column, the script simplifies the dataset and avoids introducing potential inaccuracies in subsequent analyses.

  3. Handling Missing Values in 'Embarked':
     - Rows with missing values in the 'Embarked' column are dropped from the dataset. This decision is made due to the low percentage of missing values in this column, and it ensures that the impact on the dataset's size is minimal.

  4. Final Dataset Integrity:



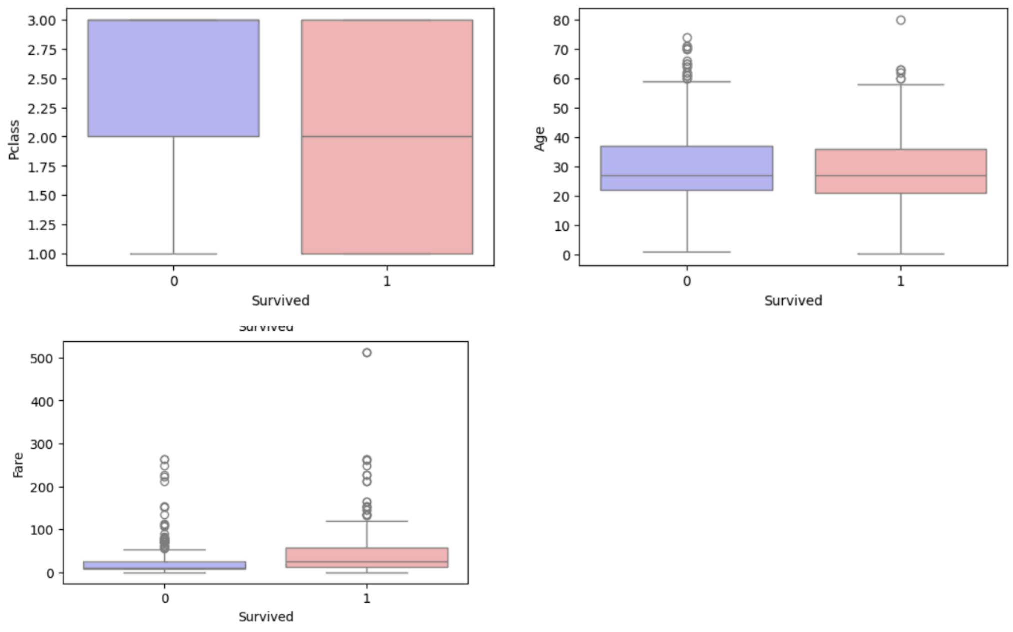
```
data.isna().sum()

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

- <u>Dropping Trivial Features</u>

  - The removal of features such as 'PassengerId', 'Name', and 'Ticket' is justified based on the understanding that these features are unlikely to significantly impact an individual's likelihood of survival.
  - This step contributes to a more focused dataset, retaining only the features deemed more relevant for subsequent analysis and modeling tasks.

- <u>Checking For Outliers</u>

  -Plots Overview:
    - The script generates box plots for each numerical feature against the 'Survived' variable, organized in a 2x2 grid.
    - Features:
      - 'P class' - A discrete feature with values 1, 2, and 3.
      - 'Age' - Age values above 60 are observed.
      - 'Fare' - Some plots show large fare values.



  - Detailed Analysis:
    - P class:
      - As 'P class' is a discrete feature with values 1, 2, and 3, no outliers are expected. The plot confirms this expectation.

    - Age:
      - The box plot for 'Age' indicates some data values above 60. However, these are not necessarily outliers; instead, they may represent elderly individuals traveling on the ship. Therefore, these values are considered valid and not treated as outliers.

- Fare:
  - The 'Fare' plot shows large values, which is reasonable for this feature, as fare prices can vary significantly. No outliers of concern are identified.

-Conclusion:
  - The analysis concludes that no outliers of significance are present in the given dataset.
  - 'Pclass' adheres to its discrete nature, 'Age' values above 60 are reasonable and not treated as outliers, and 'Fare' values, including large ones, are considered within the expected range.

- <u>Categorical Encoding</u>:

  - The goal of the code is to perform categorical encoding for certain features in the Titanic dataset, namely 'Pclass', 'Sex', and 'Embarked'.

  - 'Pclass', 'Sex', and 'Embarked' are the categorical features.

  - 'P class' contains discrete values (1, 2, 3), and encoding is not needed as these values are already numerical.

  - Encoding 'Sex' Feature:
    'Sex' is encoded by replacing "female" with 0 and "male" with 1.

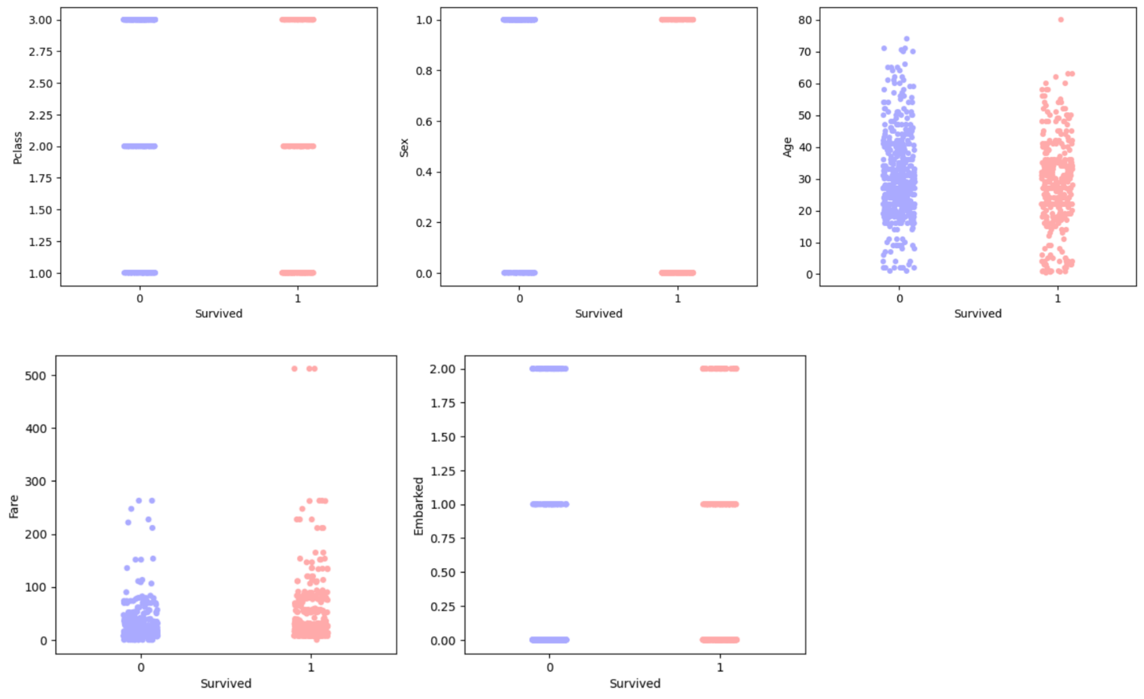
  - Encoding 'Embarked' Feature:
    A custom function, `encode_embarked`, is defined to encode 'Embarked' based on specific conditions.
    'Embarked' is encoded using the defined function, replacing 'S' with 0, 'C' with 1, and 'Q' with 2.

- <u>Visualizing Feature Target Dependence:</u>

  - The code aims to visualize the dependence of various features on the target variable 'Survived' in the Titanic dataset.

  - Plot Overview:
    - The script generates a grid of strip plots, each illustrating the relationship between a specific feature and the target variable 'Survived'.
    - Features:
      - 'Pclass'
      - 'Sex'
      - 'Age'
      - 'Fare'

- 'Embarked'



- Visual Insights:
  - 'Pclass':
    - Survival seems to be influenced by passenger class, with a higher proportion of survival in first class.

  - 'Sex':
    - The strip plot for 'Sex' indicates differences in survival rates between genders.

  - 'Age':
    - Age distribution does not show a clear trend; survival is observed across various age groups.

  - 'Fare':
    - Higher fare values appear to be associated with a higher likelihood of survival.

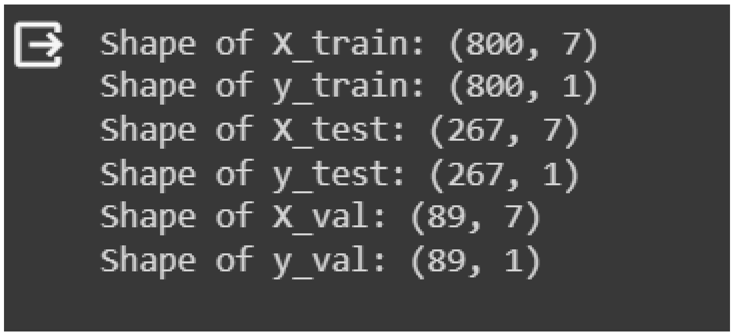
  - 'Embarked':
    - The distribution of embarkation points does not exhibit a pronounced impact on survival.

- Conclusion:
  - Features such as 'Pclass', 'Sex', 'Fare' seem to exhibit distinct patterns in relation to survival, indicating their potential importance in predictive modeling.

- <u>Splitting The Dataset</u>
  - The code is designed to split the Titanic dataset into training, testing, and validation sets,in 70-10-20 split.

  - The target variable 'Survived' is extracted into a separate DataFrame `y`.
  - The features are defined in DataFrame `X` by dropping the 'Survived' column.

  - Train-Test-Validation Split:
    - The dataset is split into training, testing, and validation sets using `train_test_split` from the scikit-learn library.



```
Shape of X_train: (800, 7)
Shape of y_train: (800, 1)
Shape of X_test: (267, 7)
Shape of y_test: (267, 1)
Shape of X_val: (89, 7)
Shape of y_val: (89, 1)
```

Task 2 Implementing entropy as the cost function to calculate split:

- calculate_entropy:
  - Calculates the entropy of a set of labels.
  - Inputs:
    - `labels`: A 1D array or list containing the target labels.
  - Outputs:
    - Returns the entropy value.

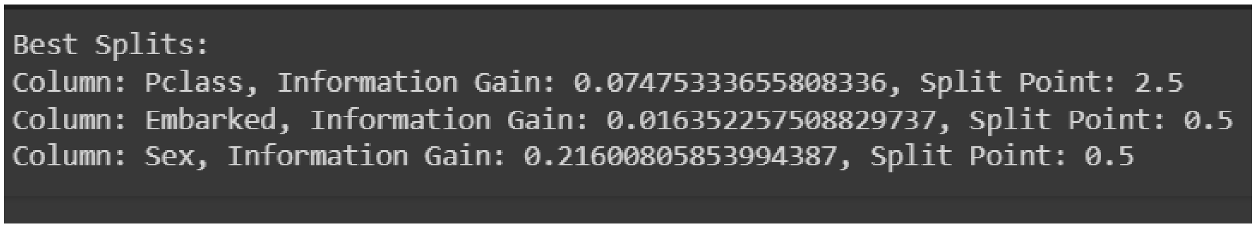
- calculate_information_gain_at_split(parent_labels, split_data, split_point):
  - Calculates the information gain at a specific split point for a given feature.
  - Inputs:
    - `parent_labels`: The target labels of the parent node.
    - `split_data`: A list containing two subsets of target labels resulting from the split.
    - `split_point`: The value at which the split is performed.
  - Outputs:
    - Returns the information gained at the split point.

- find_best_split_for_column(features, target, column):**
  - Finds the best split point for a specific column in the dataset.
  - Inputs:
    - `features`: The features (dataframe) of the dataset.
    - `target`: The target variable.
    - `column`: The column for which the best split point is to be found.
  - Outputs:
    - Returns the best information gain and the corresponding split point for the given column.

- find_best_split(features, target, columns):
  - Finds the best split points for each column specified in the `columns` list.
  - Inputs:
    - `features`: The features (dataframe) of the dataset.
    - `target`: The target variable.
    - `columns`: A list of column names for which the best split points are to be found.
  - Outputs:
    - Returns a dictionary with information gain values as keys and details (column name and split point) as values.

- best_splits = find_best_split(data, data['Survived'], selected_columns):
  - Calls the `find_best_split` function to find the best split points for the specified columns.
  - Outputs:
    - `best_splits`: A dictionary where keys are information gain values, and values are dictionaries containing column names and split points.

- The script prints the best splits, including the column name, information gain, and split point for each selected column.



```
Best Splits:
Column: Pclass, Information Gain: 0.07475333655808336, Split Point: 2.5
Column: Embarked, Information Gain: 0.016352257508829737, Split Point: 0.5
Column: Sex, Information Gain: 0.21600805853994387, Split Point: 0.5
```

Task 3 Implementing the conTocat function:

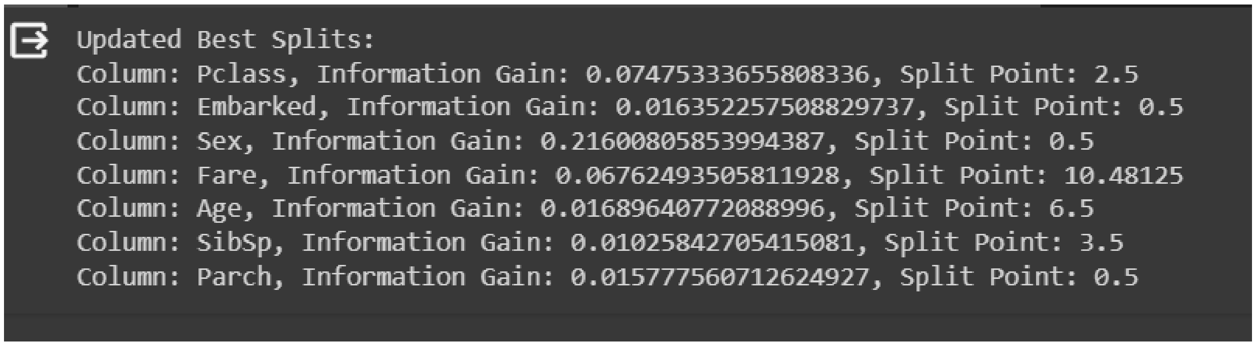
- find_best_split_at_unique_values(features, target, column, unique_values):
  - This function is introduced to handle categorical variables.
  - For each unique value in the specified column, it calculates the information gain by splitting the dataset into two subsets based on whether the value matches the current unique value.
  - The best split point is determined by selecting the unique value that maximizes the information gain.

- conTocat(features, target, column):
  - This function handles both continuous and categorical variables.
  - For continuous variables:
    - It calculates the information gain for different split points.
    - The best split point is determined based on the maximum information gain.
  - For categorical variables:
    - It utilizes the original unique values and calls `find_best_split_at_unique_values`.

- best_splits[information_gain] = {'column': column, 'split_point': split_point}

  - Updates the `best_splits` dictionary with the information gain, column name, and split point for each continuous variable.

- After processing continuous variables, the script prints the updated dictionary containing information gain, column name, and split point for all considered variables.



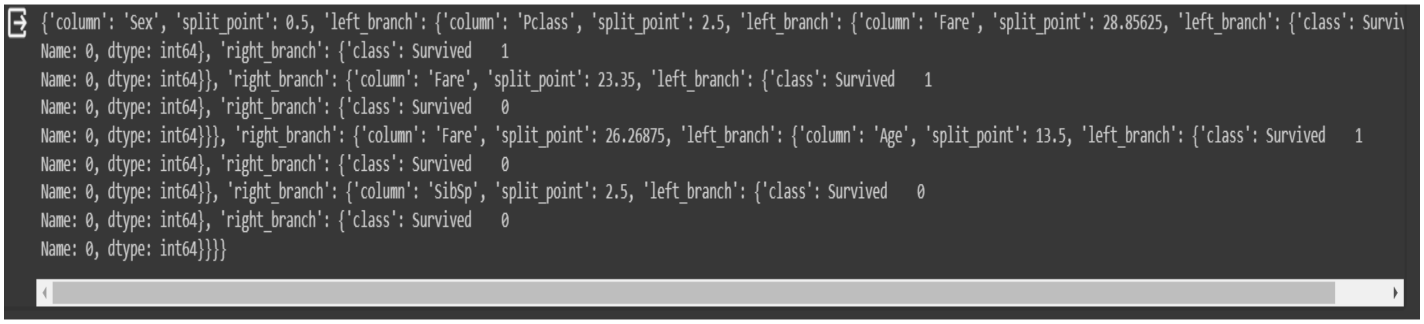
```
Updated Best Splits:
Column: Pclass, Information Gain: 0.07475333655808336, Split Point: 2.5
Column: Embarked, Information Gain: 0.016352257508829737, Split Point: 0.5
Column: Sex, Information Gain: 0.21600805853994387, Split Point: 0.5
Column: Fare, Information Gain: 0.06762493505811928, Split Point: 10.48125
Column: Age, Information Gain: 0.01689640772088996, Split Point: 6.5
Column: SibSp, Information Gain: 0.01025842705415081, Split Point: 3.5
Column: Parch, Information Gain: 0.015777560712624927, Split Point: 0.5
```

Task 4 Implementing the Train function to build the decision tree:

- The function includes a stopping criterion based on the `max_depth` parameter. If the current depth of the tree exceeds or equals the specified `max_depth`, the function returns a leaf node with the majority class as the predicted class.

- The function uses the `find_best_split` function to identify the best splits for the current dataset.
- If no valid splits are possible or the target is empty, the function returns a leaf node with the majority class as the predicted class.
- The dataset is split into left and right branches based on the best split condition.
- The function recursively calls itself to build the decision tree for each branch.

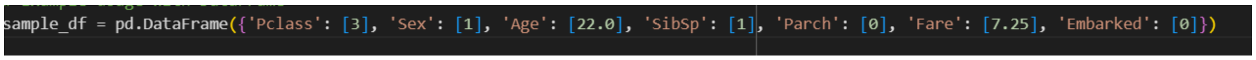
In summary, the `train_decision_tree` function recursively constructs a decision tree by identifying the best splits and splitting the dataset accordingly. The stopping criteria prevent the tree from growing beyond a specified depth, and the resulting decision tree is represented as a nested dictionary.



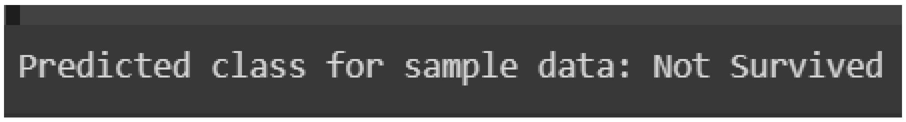
```
{'column': 'Sex', 'split_point': 0.5, 'left_branch': {'column': 'Pclass', 'split_point': 2.5, 'left_branch': {'column': 'Fare', 'split_point': 28.85625, 'left_branch': {'class': Surviv
Name: 0, dtype: int64}, 'right_branch': {'class': Survived    1
Name: 0, dtype: int64}}, 'right_branch': {'column': 'Fare', 'split_point': 23.35, 'left_branch': {'class': Survived    1
Name: 0, dtype: int64}, 'right_branch': {'class': Survived    0
Name: 0, dtype: int64}}}, 'right_branch': {'column': 'Fare', 'split_point': 26.26875, 'left_branch': {'column': 'Age', 'split_point': 13.5, 'left_branch': {'class': Survived    1
Name: 0, dtype: int64}, 'right_branch': {'class': Survived    0
Name: 0, dtype: int64}}, 'right_branch': {'column': 'SibSp', 'split_point': 2.5, 'left_branch': {'class': Survived    0
Name: 0, dtype: int64}, 'right_branch': {'class': Survived    0
Name: 0, dtype: int64}}}}
```

Task 5 Implementing the Infer function:

- The function takes two parameters: `sample_df` (a DataFrame with a single row of feature values) and `decision_tree` (the trained decision tree).
- It recursively traverses the decision tree to make a prediction for the given sample.
- If the decision tree node is a leaf node (i.e., it does not have 'left_branch' and 'right_branch' keys), the function returns the predicted class (if available) for that leaf node.
- A sample DataFrame (`sample_df`) is created with feature values for Pclass, Sex, Age, SibSp, Parch, Fare, and Embarked.
  The `infer` function is called with the sample DataFrame and the trained decision tree (
  The predicted class is then printed based on the returned value.



```
sample_df = pd.DataFrame({'Pclass': [3], 'Sex': [1], 'Age': [22.0], 'SibSp': [1], 'Parch': [0], 'Fare': [7.25], 'Embarked': [0]})
```



```
Predicted class for sample data: Not Survived
```

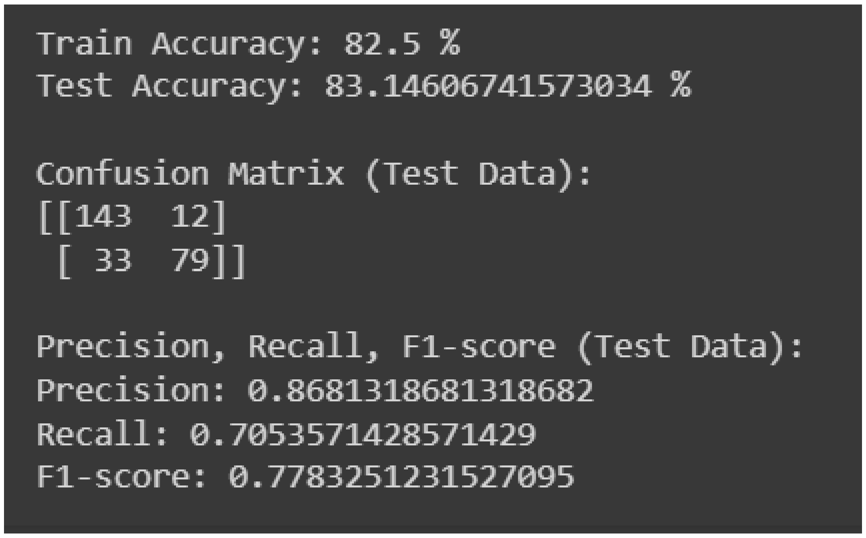
Task 6,7,8 Getting the Accuracy, Confusion Matrix, Precision, Recall and F1 Score:

- predict` Function:
  - The `predict` function takes a decision tree (`tree`) and a set of features (`features`).
  - It recursively traverses the decision tree to predict the target class for each set of features.
  - If a leaf node is reached, the predicted class is returned.

- `compute_metrics` Function:
  - The `compute_metrics` function evaluates the performance of the decision tree on a dataset.
  - It uses the `predict` function to obtain predictions for the provided features.
  - The function computes accuracy, confusion matrix, precision, recall, and F1-score using scikit-learn metrics.

- Evaluation on Training Set:
  - The code uses the `compute_metrics` function to evaluate the decision tree on the training set (`X_train` and `y_train`).
  - The results, including accuracy, confusion matrix, precision, recall, and F1-score, are stored in variables (`train_accuracy`, `train_confusion`, `train_precision`, `train_recall`, `train_f1`).

- Evaluation on Test Set:
  - Similarly, the code evaluates the decision tree on the test set (`X_test` and `y_test`).
  - The results are stored in variables (`test_accuracy`, `test_confusion`, `test_precision`, `test_recall`, `test_f1`).



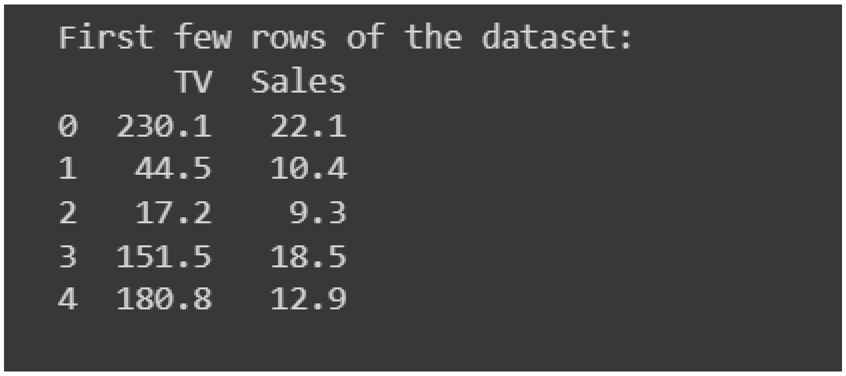
```
Train Accuracy: 82.5 %
Test Accuracy: 83.14606741573034 %

Confusion Matrix (Test Data):
[[143  12]
 [ 33  79]]

Precision, Recall, F1-score (Test Data):
Precision: 0.8681318681318682
Recall: 0.7053571428571429
F1-score: 0.7783251231527095
```

QUESTION 2

Linear Regression

Task 1 Dataset Exploration:

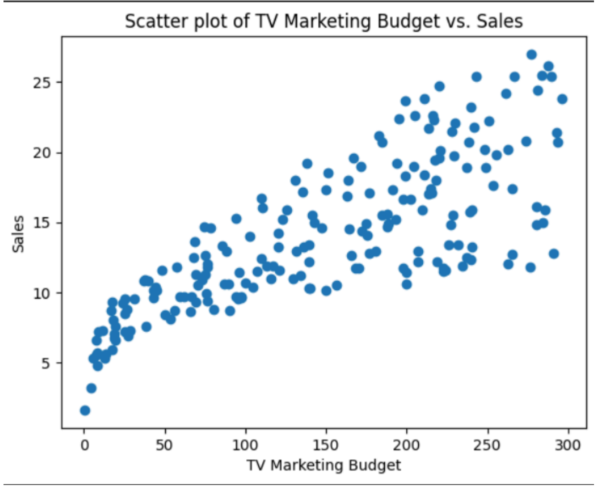
- Loading Data:



```
First few rows of the dataset:
      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
```

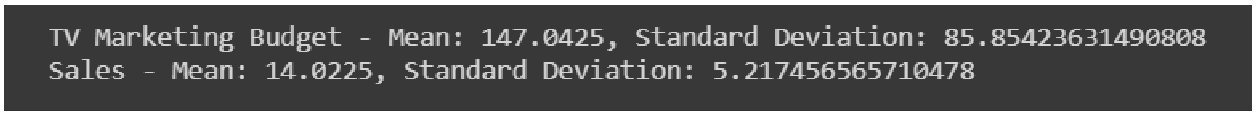
- Creating a Scatter Plot:

  - This generates a scatter plot using Matplotlib, where the x-axis represents the 'TV' column and the y-axis represents the 'Sales' column. It is a visual representation of the relationship between TV marketing budget and sales.



- Calculating Mean and Standard Deviation:

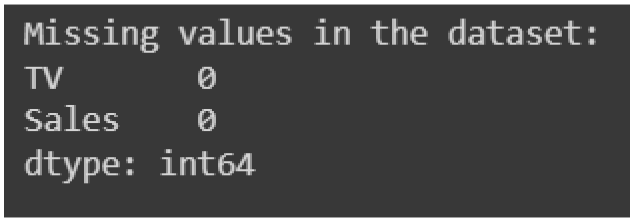
  Calculates the mean and standard deviation for both 'TV' and 'Sales' columns.



```
TV Marketing Budget - Mean: 147.0425, Standard Deviation: 85.85423631490808
Sales - Mean: 14.0225, Standard Deviation: 5.217456565710478
```

Task 2 Data Pre Processing:

- Checking for Missing Values:
  This code prints the sum of missing values for each column in the DataFrame.



```
Missing values in the dataset:
TV       0
Sales    0
dtype: int64
```

- Normalizing the Columns using StandardScaler:

  - The `StandardScaler` from scikit-learn is used to standardize the 'TV' and 'Sales' columns. Standardization involves scaling the features to have a mean of 0 and a standard deviation of 1.

- Splitting the Dataset into Training and Testing Sets:

  - This code splits the DataFrame into features (`X`) and target variable (`y`). The dataset is then split into training and testing sets using the `train_test_split` function from scikit-learn. `test_size=0.2` indicates that 20% of the data will be used for testing, and

Task 3 Linear Regression Implementation:

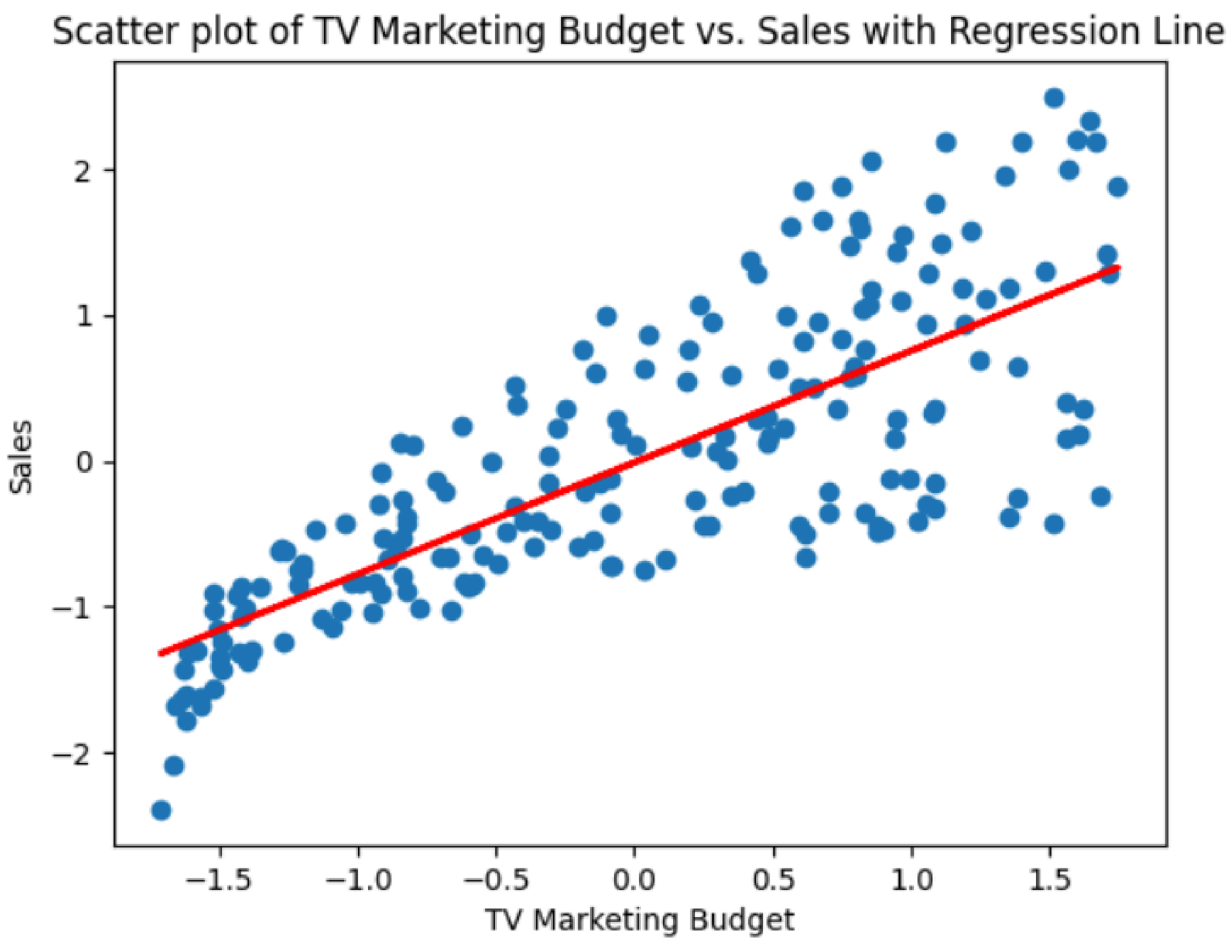
- Hypothesis Function:
  This function defines the hypothesis for linear regression, which is a simple linear equation \(h(x) = w_0 + w_1 \cdot X \).

- Cost Function (Mean Squared Error):
  The cost function calculates the mean squared error (MSE) for the given hypothesis and actual target values.

- Gradient Descent:
  The gradient descent function is used to optimize the parameters \(w_0\) and \(w_1\) by iteratively updating them in the direction that minimizes the cost function.

- Initial Values and Training:
  Sets initial values for \(w_0\) and \(w_1\), specifies the learning rate and number of epochs, and then calls the gradient descent function to train the model.

- Plotting Regression Line on Scatter Plot
  This code uses Matplotlib to create a scatter plot of the data and overlays the regression line based on the trained parameters \(w_0\) and \(w_1\).



As TV Marketing budget increases, Sales also increases Hence The linear regression is Positive.

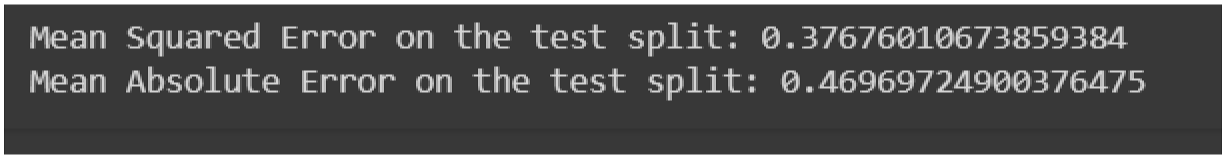
Task 4 Evaluation

- Making Predictions
  The trained linear regression model is used to make predictions on the test set. `X_test['TV']` represents the 'TV' feature of the test set, and `y_pred` will contain the predicted values.

- Calculating Mean Squared Error (MSE) and Mean Absolute Error (MAE):
  - The `mean_squared_error` function from scikit-learn is used to calculate the MSE between the actual target values (`y_test`) and the predicted values (`y_pred`).
  - The `mean_absolute_error` function is used to calculate the MAE between the actual and predicted values.

`

- The calculated MSE and MAE are then printed to the console to provide an indication of how well the model is performing on the unseen test data.



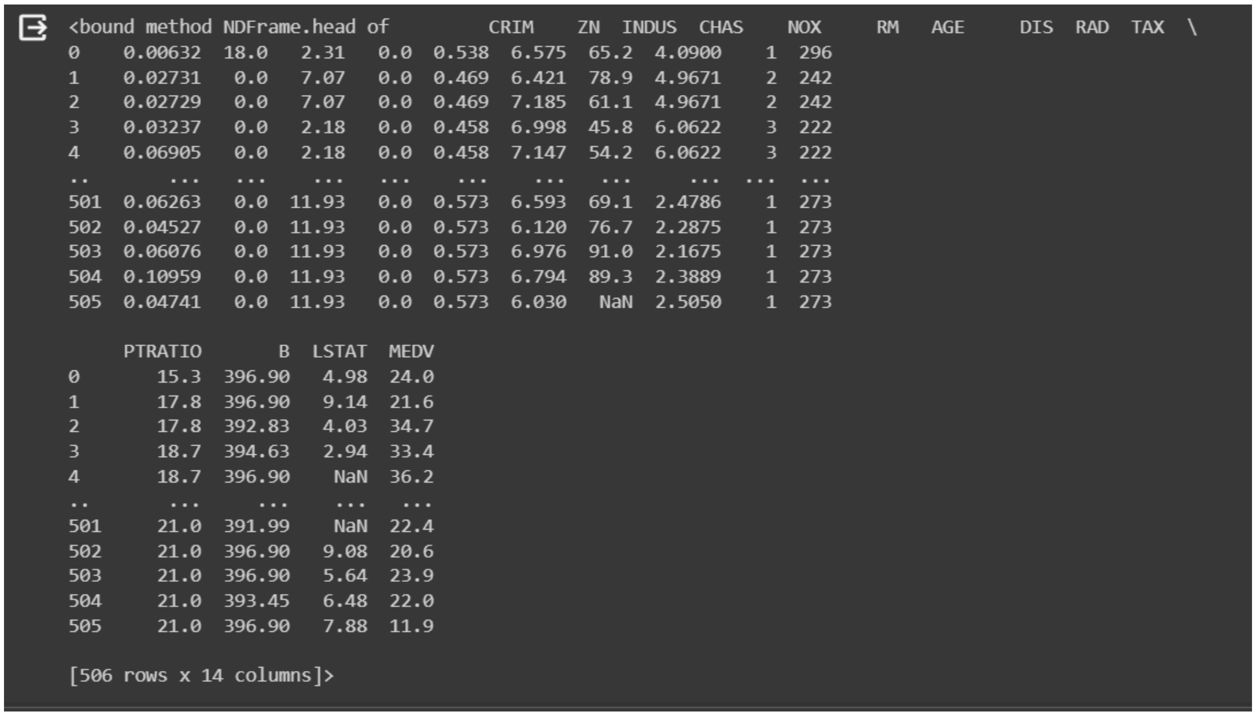
```
Mean Squared Error on the test split: 0.37676010673859384
Mean Absolute Error on the test split: 0.46969724900376475
```

QUESTION 3

Multivariate Linear Regression

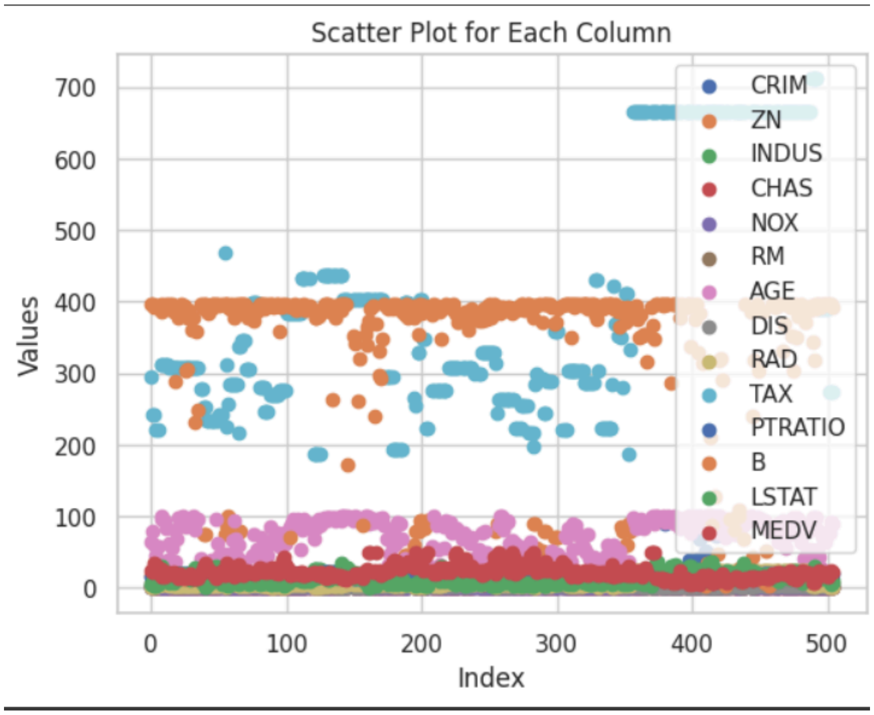
Task 1 Dataset Exploration:

- **Loading the dataset**

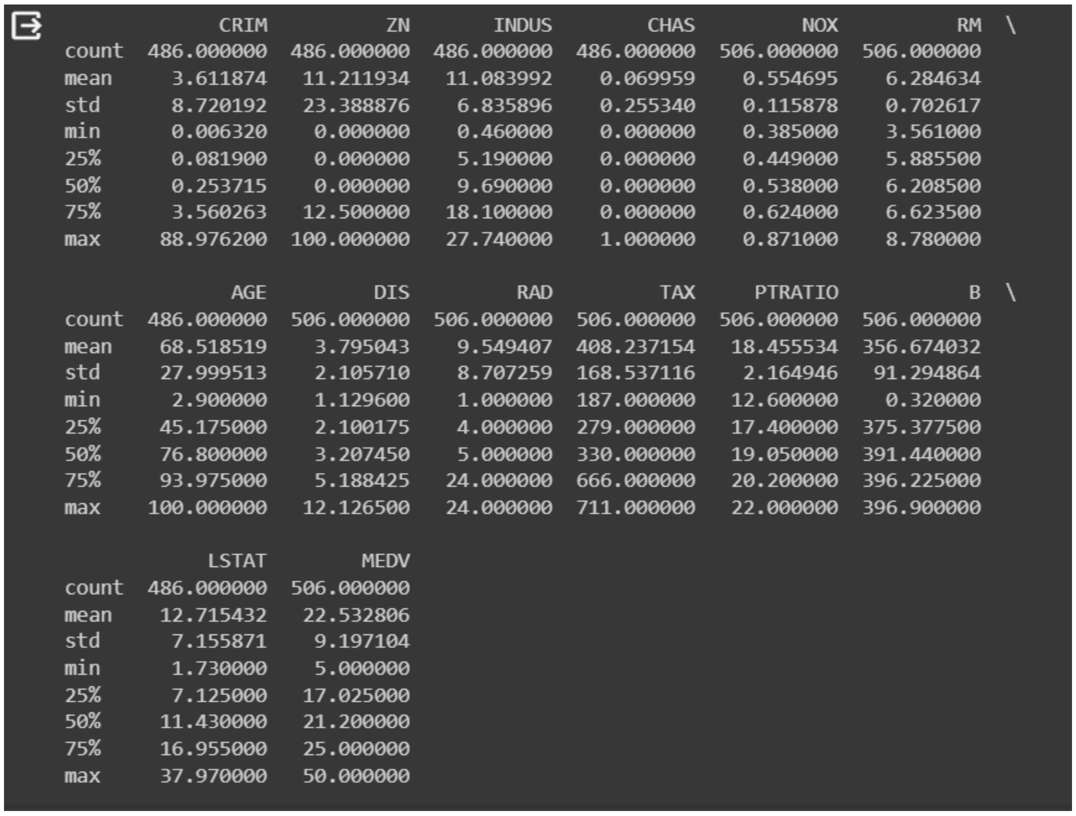


```
<bound method NDFrame.head of         CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  \
0    0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900    1  296
1    0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622    3  222
..       ...   ...    ...   ...    ...    ...   ...     ...  ...  ...
501  0.06263   0.0  11.93   0.0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93   0.0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93   0.0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93   0.0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93   0.0  0.573  6.030   NaN  2.5050    1  273

     PTRATIO       B  LSTAT  MEDV
0       15.3  396.90   4.98  24.0
1       17.8  396.90   9.14  21.6
2       17.8  392.83   4.03  34.7
3       18.7  394.63   2.94  33.4
4       18.7  396.90    NaN  36.2
..       ...     ...    ...   ...
501     21.0  391.99    NaN  22.4
502     21.0  396.90   9.08  20.6
503     21.0  396.90   5.64  23.9
504     21.0  393.45   6.48  22.0
505     21.0  396.90   7.88  11.9

[506 rows x 14 columns]>
```

- Scatter Plots:
  - A loop iterates over each column in the cleaned DataFrame (`df`).
  - For each column, a scatter plot is created using the Matplotlib library.
  - The x-axis values are the index values of the DataFrame, and the y-axis values are the corresponding values in the current column.
  - Each scatter plot is labeled with the column name for reference in the legend.



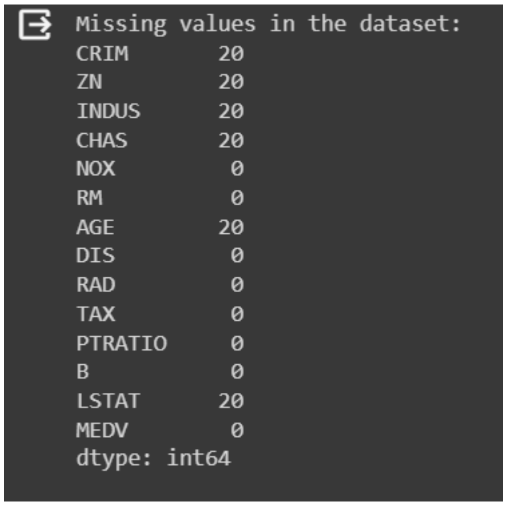
- Summary Statistics:
   - Descriptive statistics (mean, standard deviation, min, 25th percentile, median, 75th percentile, and max) for the original DataFrame (`raw_df`) are calculated using the `describe()` function.
   - The resulting statistics are stored in the variable `statistics`.



```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  486.000000  486.000000  486.000000  486.000000  506.000000  506.000000
mean     3.611874   11.211934   11.083992    0.069959    0.554695    6.284634
std      8.720192   23.388876    6.835896    0.255340    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.081900    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.253715    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.560263   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  486.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.518519    3.795043    9.549407  408.237154   18.455534  356.674032
std     27.999513    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.175000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     76.800000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     93.975000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT        MEDV
count  486.000000  506.000000
mean    12.715432   22.532806
std      7.155871    9.197104
min      1.730000    5.000000
25%      7.125000   17.025000
50%     11.430000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

Task 2 Data Pre Processing:

- Missing Values Handling:
  - The code starts by calculating the number of missing values for each column in the original DataFrame (`raw_df`). This information is stored in the `missing_values` variable.

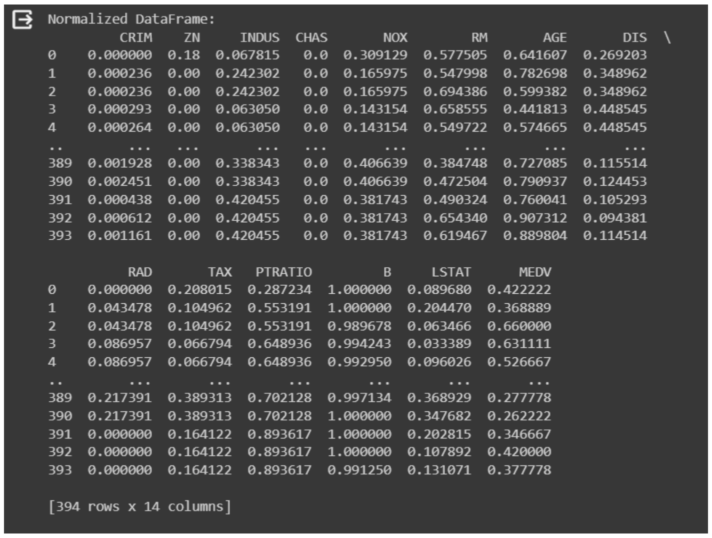


```
Missing values in the dataset:
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
MEDV        0
dtype: int64
```

- Missing values in the original DataFrame (`raw_df`) are filled using the mean of each respective column. This helps in preserving the overall distribution while providing a reasonable estimate for missing entries.

- Normalization:

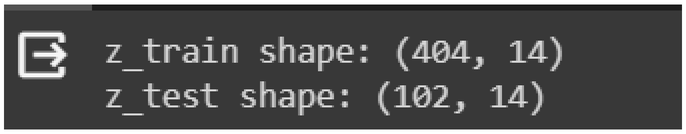
- The script utilizes the Min-Max scaling technique to normalize the dataset. The `MinMaxScaler` from scikit-learn is employed to transform the values of each column to a range between 0 and 1. The normalized DataFrame is stored in `df_normalized`.



```
Normalized DataFrame:
         CRIM    ZN     INDUS  CHAS       NOX        RM       AGE       DIS  \
0    0.000000  0.18  0.067815   0.0  0.309129  0.577505  0.641607  0.269203
1    0.000236  0.00  0.242302   0.0  0.165975  0.547998  0.782698  0.348962
2    0.000236  0.00  0.242302   0.0  0.165975  0.694386  0.599382  0.348962
3    0.000293  0.00  0.063050   0.0  0.143154  0.658555  0.441813  0.448545
4    0.000264  0.00  0.063050   0.0  0.143154  0.549722  0.574665  0.448545
..        ...   ...       ...   ...       ...       ...       ...       ...
389  0.001928  0.00  0.338343   0.0  0.406639  0.384748  0.727085  0.115514
390  0.002451  0.00  0.338343   0.0  0.406639  0.472504  0.790937  0.124453
391  0.000438  0.00  0.420455   0.0  0.381743  0.490324  0.760041  0.105293
392  0.000612  0.00  0.420455   0.0  0.381743  0.654340  0.907312  0.094381
393  0.001161  0.00  0.420455   0.0  0.381743  0.619467  0.889804  0.114514

          RAD       TAX   PTRATIO         B     LSTAT      MEDV
0    0.000000  0.208015  0.287234  1.000000  0.089680  0.422222
1    0.043478  0.104962  0.553191  1.000000  0.204470  0.368889
2    0.043478  0.104962  0.553191  0.989678  0.063466  0.660000
3    0.086957  0.066794  0.648936  0.994243  0.033389  0.631111
4    0.086957  0.066794  0.648936  0.992950  0.096026  0.526667
..        ...       ...       ...       ...       ...       ...
389  0.217391  0.389313  0.702128  0.997134  0.368929  0.277778
390  0.217391  0.389313  0.702128  1.000000  0.347682  0.262222
391  0.000000  0.164122  0.893617  1.000000  0.202815  0.346667
392  0.000000  0.164122  0.893617  1.000000  0.107892  0.420000
393  0.000000  0.164122  0.893617  0.991250  0.131071  0.377778

[394 rows x 14 columns]
```

- Data Splitting:

  - The original DataFrame (`raw_df`) is split into training and testing sets using the `train_test_split` function from scikit-learn. The `z_train` and `z_test` variables store the training and testing sets, respectively.(80-20 split)



```
z_train shape: (404, 14)
z_test shape: (102, 14)
```

Task 3 Linear Regression Implementation:

- Hypothesis Function:

  The multivariate hypothesis function is defined as a linear combination of the input features, weights, and bias:

- Cost Function:

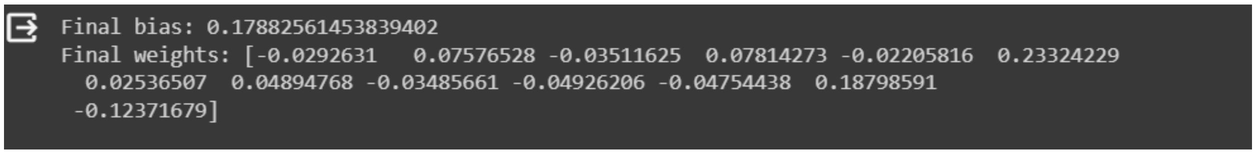
  The cost function measures the error between predicted and actual values:

- Gradient Descent:

  Gradient descent is used to minimize the cost function by updating weights and bias iteratively.

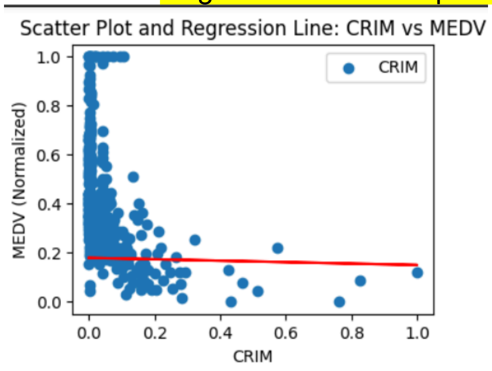
- Results and Visualization:

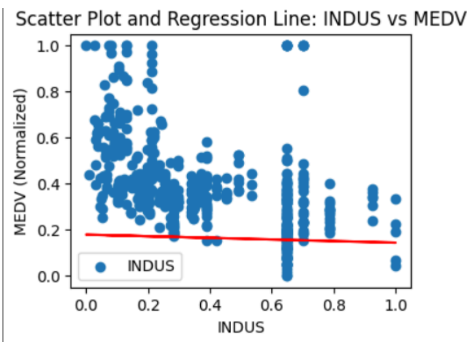
  The final bias and weights are printed, and regression

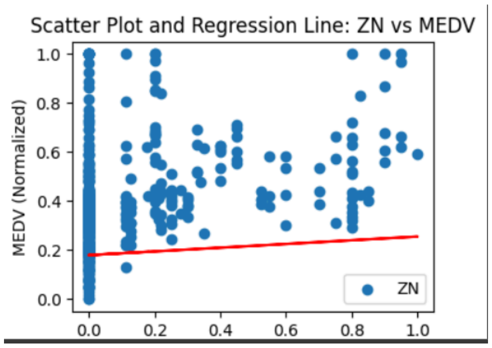


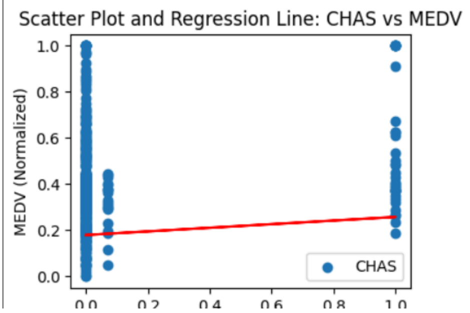
```
Final bias: 0.17882561453839402
Final weights: [-0.0292631   0.07576528 -0.03511625  0.07814273 -0.02205816  0.23324229
  0.02536507  0.04894768 -0.03485661 -0.04926206 -0.04754438  0.18798591
 -0.12371679]
```

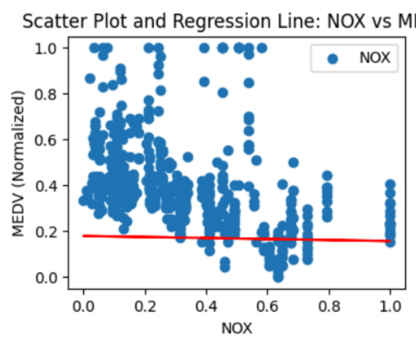
Regression lines are plotted for each feature against the target MEDV

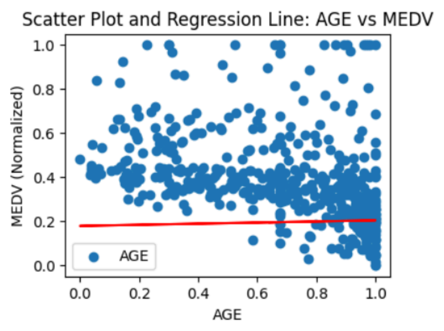


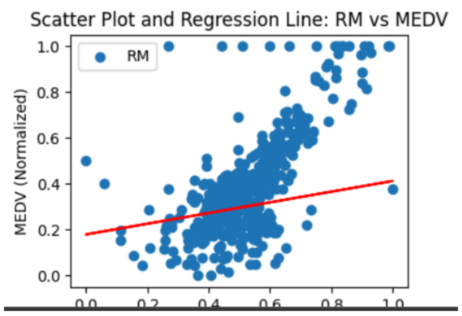


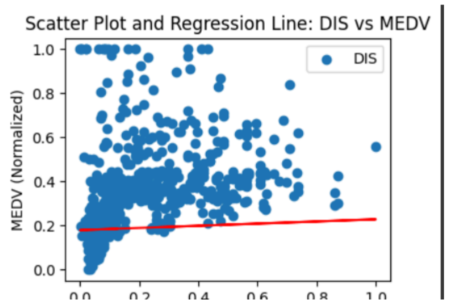


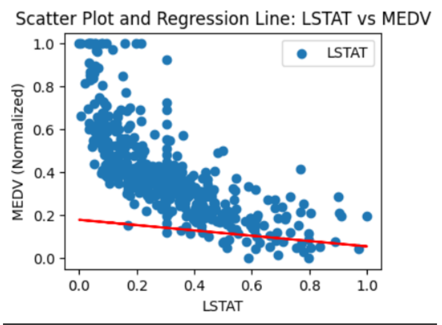


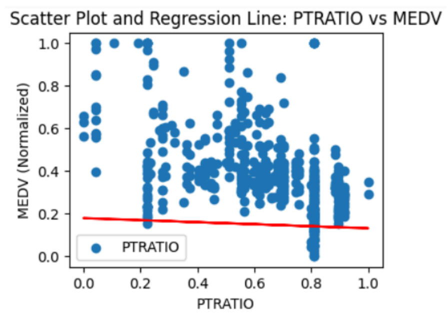

Scatter Plot and Regression Line: NOX vs MEDV
MEDV (Normalized)
NOX
NOX
0.0
0.2
0.4
0.6
0.8
1.0

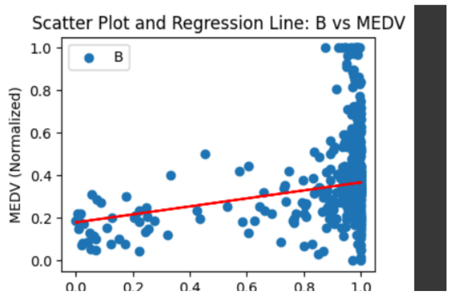

Scatter Plot and Regression Line: AGE vs MEDV
MEDV (Normalized)
AGE
AGE
0.0
0.2
0.4
0.6
0.8
1.0

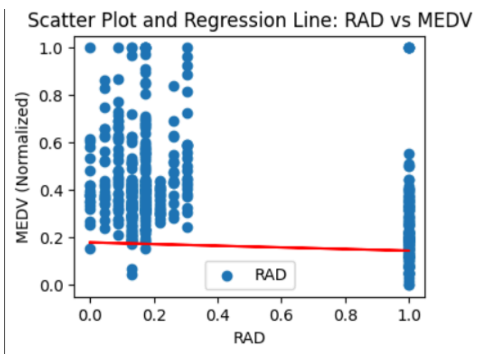

Scatter Plot and Regression Line: RM vs MEDV
MEDV (Normalized)
RM
0.0
0.2
0.4
0.6
0.8
1.0

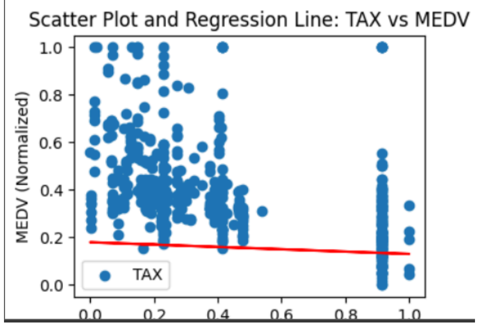

Scatter Plot and Regression Line: DIS vs MEDV
MEDV (Normalized)
DIS
0.0
0.2
0.4
0.6
0.8
1.0


Scatter Plot and Regression Line: LSTAT vs MEDV
MEDV (Normalized)
LSTAT
LSTAT
0.0
0.2
0.4
0.6
0.8
1.0


Scatter Plot and Regression Line: PTRATIO vs MEDV
MEDV (Normalized)
PTRATIO
PTRATIO
0.0
0.2
0.4
0.6
0.8
1.0


Scatter Plot and Regression Line: B vs MEDV
MEDV (Normalized)
B
0.0
0.2
0.4
0.6
0.8
1.0

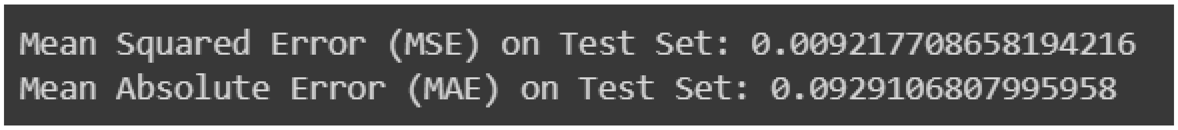

Scatter Plot and Regression Line: RAD vs MEDV
MEDV (Normalized)
RAD
RAD
0.0
0.2
0.4
0.6
0.8
1.0


Scatter Plot and Regression Line: TAX vs MEDV
MEDV (Normalized)
TAX
0.0
0.2
0.4
0.6
0.8
1.0

Task 4 Evaluation
The trained model is used to make predictions on the test set.

- MSE measures the average squared difference between predicted and actual values. It is a commonly used metric to assess the overall model performance.

- MAE represents the average absolute difference between predicted and actual values. It provides a measure of the magnitude of errors without considering their direction.



```
Mean Squared Error (MSE) on Test Set: 0.009217708658194216
Mean Absolute Error (MAE) on Test Set: 0.0929106807995958
```