Report on Lab-3 & Lab-4: CSL2050 - Pattern Recognition and Machine Learning

Submitted by: Souvik Maji (B22CS089)

Problem 1:

Task-1: Pre-processing and Visualization

Dataset Exploration:

.

- Displayed the first few rows to understand the structure of the dataset.
- Visualized the data using plots to understand the distribution of features.
- Pre-processing: Checked for missing values and handled them appropriately (if any).
- Identified the types of features (ordinal, nominal, categorical).
- Applied categorical encoding where applicable.
- Split the data into train, validation, and test sets using a 70-20-10 split.

Question 1

```
[85] import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns

     from sklearn.model_selection import train_test_split
     from sklearn import preprocessing
     from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
```

TASK-1

```
url_1 = '/content/titanic.csv'
data_1 = pd.read_csv(url_1)
data_1
```

	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
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

```
[102] print("Column 'Sex' before encoding:")
      print(data_1["Sex"][0:5])

      data_1["Sex"].replace("female", 0, inplace = True)
      data_1["Sex"].replace("male", 1, inplace = True)

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

```
[103] print("Column 'Sex' after encoding:")
      print(data_1["Sex"][0:5])

      Column 'Sex' after encoding:
      0    1
      1    0
      2    0
      3    0
      4    1
      Name: Sex, dtype: int64
```

```
def encode_(column):

  for data in column:
    if data == 'S':
      column[column.index(data)] = 0
    elif data == 'C':
      column[column.index(data)] = 1
    else:
      column[column.index(data)] = 2

  return column

print("Column 'Embarked' before encoding:")
print(data_1["Embarked"][0:5])

data_1["Embarked"] = encode_(list(data_1["Embarked"]))
```

```
[107] X_titanic = X_titanic.iloc[:, :].values
      y_titanic = y_titanic.iloc[:].values.reshape(-1,1)
      X_titanic

      array([[ 3.    , 1.    , 22.    , ..., 0.    ,  7.25  , 0.    ],
             [ 1.    , 0.    , 38.    , ..., 0.    , 71.2833, 1.    ],
             [ 3.    , 0.    , 26.    , ..., 0.    ,  7.925 , 0.    ],
             ...,
             [ 3.    , 0.    , 22.    , ..., 2.    , 23.45  , 0.    ],
             [ 1.    , 1.    , 26.    , ..., 0.    , 30.    , 1.    ],
             [ 3.    , 1.    , 32.    , ..., 0.    ,  7.75  , 2.    ]])
```

Task-2

```
[108] X_train_3, X_val_test_3, y_train_3, y_val_test_3 = train_test_split(X_titanic, y_titanic, train_size=.7, random_state=41)
      X_val_3, X_test_3 , y_val_3, y_test_3 = train_test_split(X_val_test_3, y_val_test_3, test_size=.333, random_state=41)
```

Task-2: Entropy as Cost Function

- Implemented entropy as the cost function to calculate the split.

```
[109] def information_gain(self, parent, l_child, r_child, mode="entropy"):
        # function to compute information gain

        weight_l = len(l_child) / len(parent)
        weight_r = len(r_child) / len(parent)
        if mode=="gini":
            gain = self.gini_index(parent) - (weight_l*self.gini_index(l_child) + weight_r*self.gini_index(r_child))
        else:
            gain = self.entropy(parent) - (weight_l*self.entropy(l_child) + weight_r*self.entropy(r_child))
        return gain

    def entropy(self, y):
        # function to compute entropy

        class_labels = np.unique(y)
        entropy = 0
        for cls in class_labels:
            p_cls = len(y[y == cls]) / len(y)
            entropy += -p_cls * np.log2(p_cls)
        return entropy
```

Task-3: Converting Continuous Variables to Categorical

- Implemented the conTocat() function to convert continuous variables to categorical.
- Ensured that continuous variables are independent of each other for the split.

```
def conTocat(Y, threshold_value, column_name):
    X = Y.copy(deep=True)
    X[column_name] = X[column_name].apply(lambda x : 1 if x >= threshold_value  else 0)
    return X

def infer(self, x, tree):
    # function to predict a single data point

    if tree.value!=None: return tree.value
    feature_val = x[tree.feature_index]
    if feature_val<=tree.threshold:
      return self.infer(x, tree.left)
    else:
      return self.infer(x, tree.right)
```

Task-4: Training Function Implemented the training function for the decision tree. Developed helper functions to:

- Get the attribute leading to the best split.
- Make the split based on the selected attribute.
- Self-identify when there is no information gain during training.
- Incorporated properties like max depth to control the tree's growth.

```python
class Node():
    def __init__(self, feature_index=None, threshold=None, left=None, right=None, info_gain=None, value=None):

        # for decision node
        self.feature_index = feature_index
        self.threshold = threshold
        self.left = left
        self.right = right
        self.info_gain = info_gain

        # for leaf node
        self.value = value
```

```python
class DecisionTreeClassifier():
    def __init__(self, min_samples_split=2, max_depth=2):


        # initialize the root of the tree
        self.root = None

        # stopping conditions
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth

    def build_tree(self, dataset, curr_depth=0):


        X, Y = dataset[:,:-1], dataset[:,-1]
        num_samples, num_features = np.shape(X)

        # split until stopping conditions are met
        if num_samples>=self.min_samples_split and curr_depth<=self.max_depth:
            # find the best split
            best_split = self.get_best_split(dataset, num_samples, num_features)
            # check if information gain is positive
            if best_split["info_gain"]>0:
                # recur left
                left_subtree = self.build_tree(best_split["dataset_left"], curr_depth+1)
                # recur right
                right_subtree = self.build_tree(best_split["dataset_right"], curr_depth+1)
                # return decision node
                return Node(best_split["feature_index"], best_split["threshold"],
                            left_subtree, right_subtree, best_split["info_gain"])

        # compute leaf node
        leaf_value = self.calculate_leaf_value(Y)
        # return leaf node
        return Node(value=leaf_value)

    def get_best_split(self, dataset, num_samples, num_features):
        # function to find the best split
```

Task-5: Inference Function

- Implemented the Infer function to classify a sample using the decision tree.

```python
def infer(self, x, tree):
# function to predict a single data point

    if tree.value!=None: return tree.value
    feature_val = x[tree.feature_index]
    if feature_val<=tree.threshold:
        return self.infer(x, tree.left)
    else:
        return self.infer(x, tree.right)
```

Task-6: Accuracy Calculation
• Computed the accuracy on the training and test splits.

TASK-6

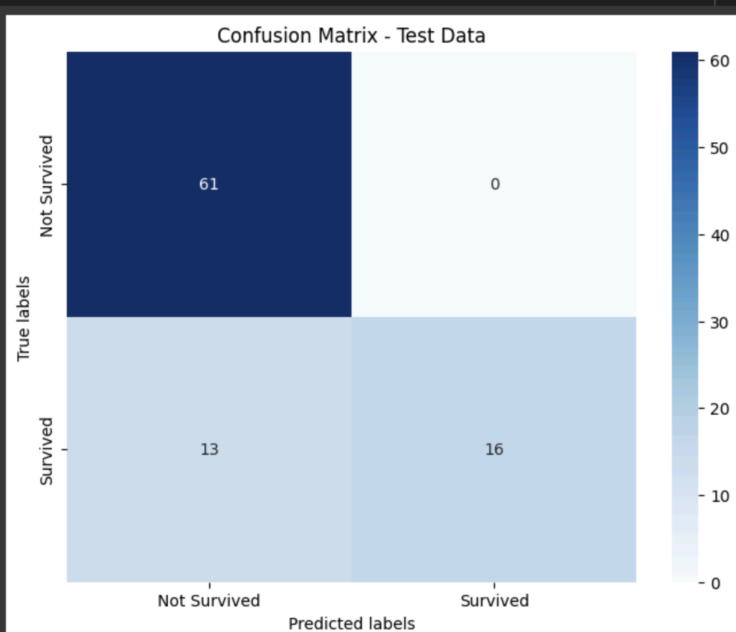
```python
y_pred_3 = classifier.predict(X_test_3)
from sklearn.metrics import accuracy_score
accuracy_score(y_test_3, y_pred_3)
```

```
0.8555555555555555
```

Task-7: Confusion Matrix

- Displayed the confusion matrix on the test data.

```python
# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Survived', 'Survived'], yticklabels=['Not Survived', 'Survived'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix - Test Data')
plt.show()
```



Task-8: Precision, Recall, F1-score Computed precision, recall, and F1-score of the Decision Tree on the test split.

TASK-8

```python
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test_3, y_pred_3)
recall = recall_score(y_test_3, y_pred_3)
f1 = f1_score(y_test_3, y_pred_3)

# Print the results
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

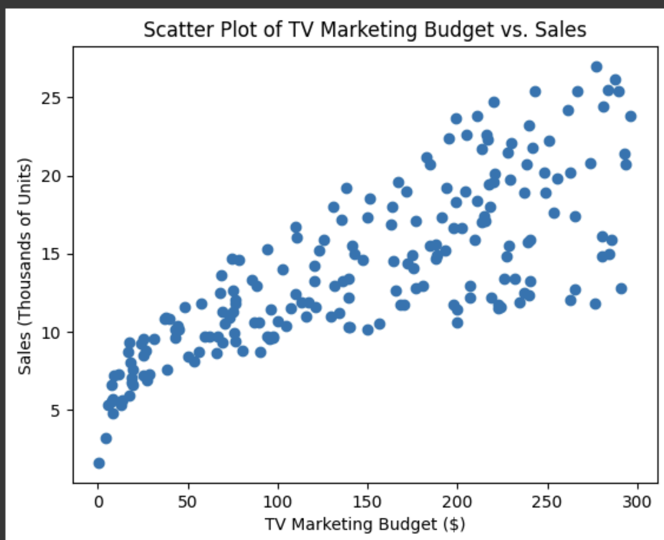
```
Precision: 1.0
Recall: 0.5517241379310345
F1-score: 0.7111111111111111
```

Problem 2: Linear Regression-1

Task-1: Dataset Exploration

- Loaded the dataset containing TV marketing budget and sales data.
- Visualized the relationship between the TV marketing budget and sales using a scatter plot.

```
# Plotting the scatter plot
plt.scatter(data['TV'], data['Sales'])
plt.xlabel('TV Marketing Budget ($)')
plt.ylabel('Sales (Thousands of Units)')
plt.title('Scatter Plot of TV Marketing Budget vs. Sales')
plt.show()
```




```
[123] # Calculating basic statistical measures
      print("TV Marketing Budget - Mean:", data['TV'].mean(), "Standard Deviation:", data['TV'].std())
      print("Sales - Mean:", data['Sales'].mean(), "Standard Deviation:", data['Sales'].std())

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

Task-2: Data Preprocessing

- Checked for missing values and handled them if found.
- Normalized the TV marketing budget and sales columns if needed.
- Split the dataset into training and testing sets using an 80-20 split.

Task-3: Linear Regression Implementation

- Implemented the hypothesis function for linear regression (y = w1x + w0) using Gradient Descent.
- Used mean squared error (MSE) as the cost function.
- Plotted the regression line on the scatter plot from Task-1.

```python
import numpy as np

# Hypothesis function: h(x) = w1*x + w0
def hypothesis(x, w0, w1):
    return w0 + w1*x

# Mean Squared Error cost function
def cost_function(X, y, w0, w1):
    m = len(y)
    cost = sum([(hypothesis(X[i], w0, w1) - y[i])**2 for i in range(m)]) / (2*m)
    return cost

X_train_np = X_train.squeeze().values
y_train_np = y_train.values

# Gradient Descent to update parameters w0 and w1
def gradient_descent(X, y, w0, w1, learning_rate, iterations, tolerance):
    m = len(y)
    cost_history = [0] * iterations

    for iteration in range(iterations):
        y_pred = hypothesis(X, w0, w1)
        loss = y_pred - y

        w0_gradient = np.sum(loss) / m
        w1_gradient = np.sum(loss * X) / m

        w0 = w0 - (learning_rate * w0_gradient)
        w1 = w1 - (learning_rate * w1_gradient)

        cost = cost_function(X, y, w0, w1)
        cost_history[iteration] = cost

        # If the cost change is less than the tolerance, break out of the loop
        if iteration > 0 and abs(cost_history[iteration-1] - cost) < tolerance:
            break

    return w0, w1, cost_history[iteration]
```

```python
plt.scatter(X_train, y_train, color='blue', label='Data Points')
plt.plot(X_train, hypothesis(X_train_np, w0, w1), color='red', label='Regression Line')
plt.xlabel('TV Marketing Budget ($)')
plt.ylabel('Sales (Thousands of Units)')
plt.title('Linear Regression Fit')
plt.legend()
plt.show()
```

Task-4: Evaluation

- On the test split, computed mean square error and absolute error for evaluation.

```python
[129] from sklearn.metrics import mean_squared_error, mean_absolute_error

y_pred = hypothesis(X_test.squeeze().values, w0, w1)


mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f"Mean Squared Error on test set: {mse}")
print(f"Mean Absolute Error on test set: {mae}")
```

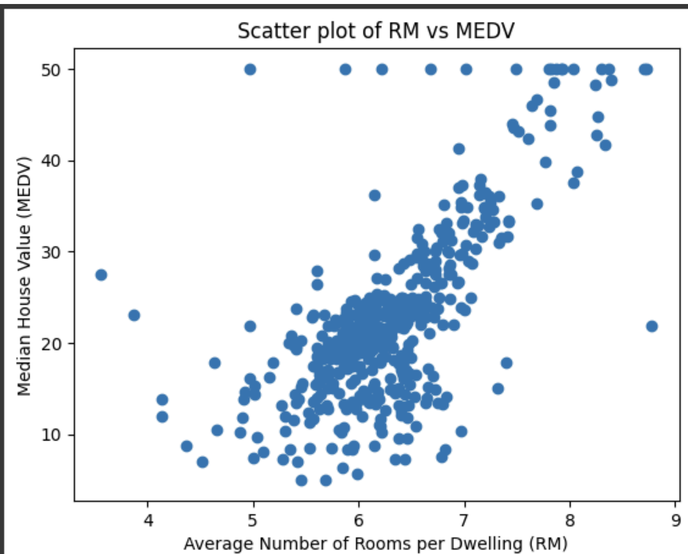
```
Mean Squared Error on test set: 0.016174951651880896
Mean Absolute Error on test set: 0.09873417835977978
```

Problem 3: Linear Regression-2

- Data Loading and Preprocessing

```python
# Load the Boston Housing dataset

boston_data = pd.read_csv("/content/bostonHousingData.csv")
column_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']

# Assign column names
boston_data.columns = column_names

# Display the first few rows
print(boston_data.head())

# Plot a scatter plot to visualize the relationship between a feature (e.g., RM) and house rent (MEDV)
plt.scatter(boston_data['RM'], boston_data['MEDV'])
plt.title('Scatter plot of RM vs MEDV')
plt.xlabel('Average Number of Rooms per Dwelling (RM)')
plt.ylabel('Median House Value (MEDV)')
plt.show()

# Calculate and display basic statistical measures
mean_rm = boston_data['RM'].mean()
std_rm = boston_data['RM'].std()
mean_medv = boston_data['MEDV'].mean()
std_medv = boston_data['MEDV'].std()

print(f'Mean RM: {mean_rm}, Standard Deviation RM: {std_rm}')
print(f'Mean MEDV: {mean_medv}, Standard Deviation MEDV: {std_medv}')
```

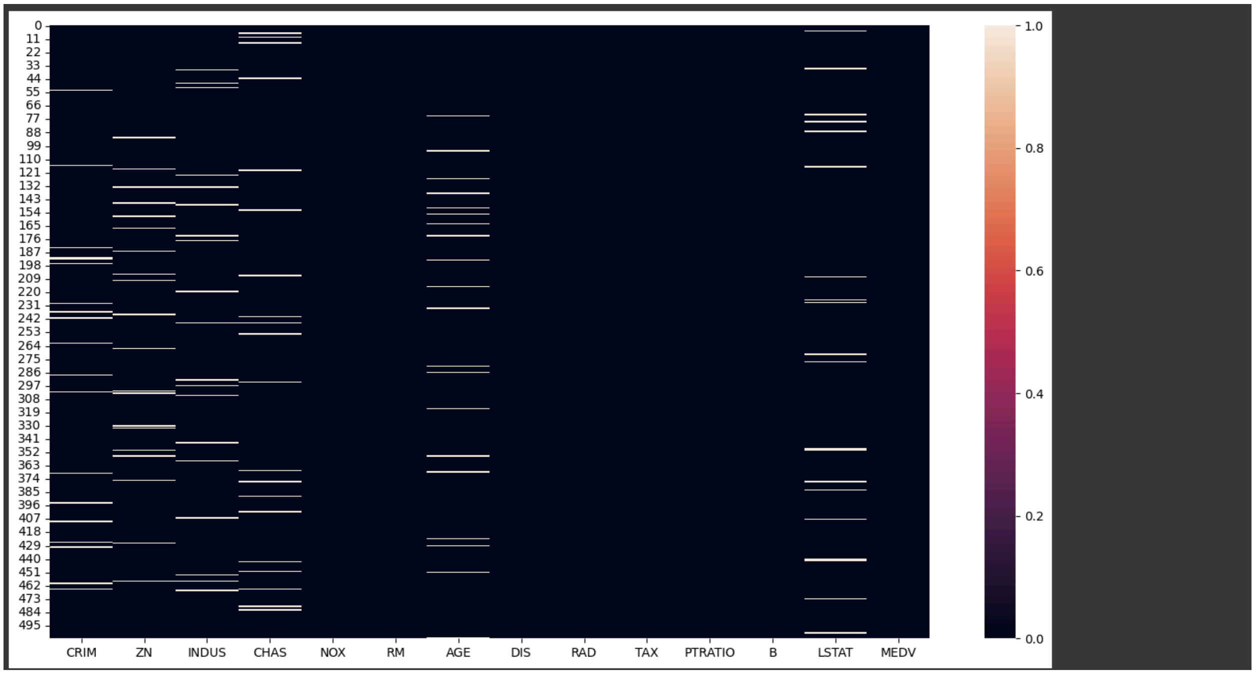
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  PTRATIO  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622    3  222     18.7

        B  LSTAT  MEDV
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90    NaN  36.2
```




```
Mean RM: 6.284634387351779, Standard Deviation RM: 0.7026171434153233
Mean MEDV: 22.532806324110677, Standard Deviation MEDV: 9.197104087379818
```

- Reads a CSV file named "bostonHousingData.csv" into a Pandas DataFrame.
- Drops the first row of the DataFrame, which presumably contains column names.

- Replaces missing values in each column with the mean of that column using fillna.




```
[146] #no columns dropped till now because none of the columns have unknown values greater than 50%
      #fill missing positions with column mean
      #boston_data['CRIM '] = boston_data['CRIM '].fillna(boston_data['CRIM '].mean())
      boston_data['ZN'] = boston_data['ZN'].fillna(boston_data['ZN'].mean())
      boston_data['INDUS'] = boston_data['INDUS'].fillna(boston_data['INDUS'].mean())
      boston_data['CHAS'] = boston_data['CHAS'].fillna(boston_data['CHAS'].mean())
      boston_data['AGE'] = boston_data['AGE'].fillna(boston_data['AGE'].mean())
      #boston_data['LSTAT'] = boston_data['LSTAT'].fillna(boston_data['LSTAT '].mean())
      print(boston_data.columns[boston_data.isnull().any()])

      Index(['CRIM', 'LSTAT'], dtype='object')

[147] len(boston_data.columns[boston_data.isnull().any()])

      2
```

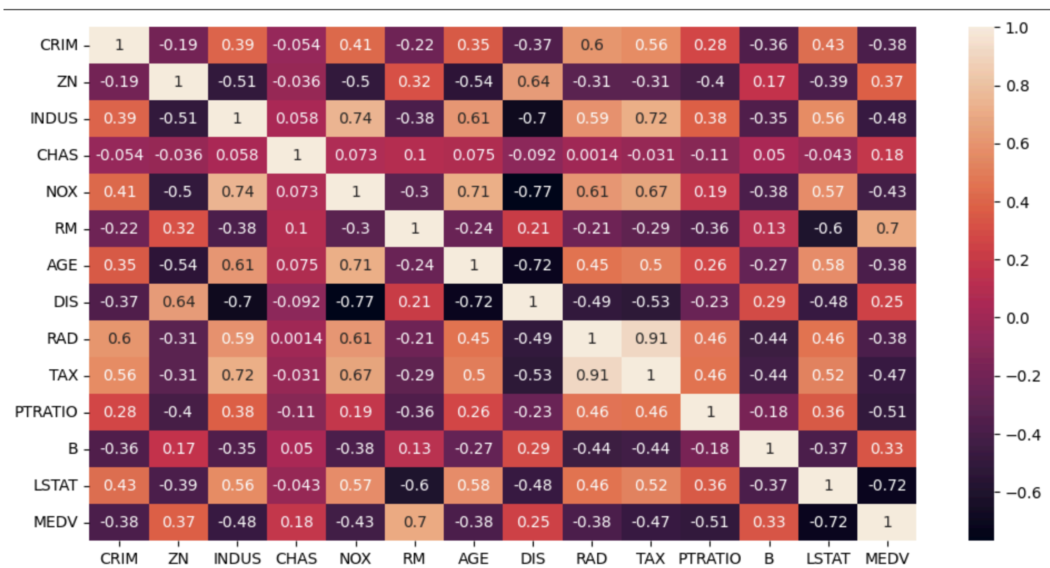
```
[149] boston_data['CRIM'] = boston_data['CRIM'].fillna(boston_data['CRIM'].mode()[0])
      boston_data['LSTAT'] = boston_data['LSTAT'].fillna(boston_data['LSTAT'].mode()[0])

[150] len(boston_data.columns[boston_data.isnull().any()])

      0
```

- Data Normalization

```python
from sklearn.preprocessing import MinMaxScaler


# Columns to normalize
columns_after_normalization = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'MEDV']
scaler = MinMaxScaler()
# Apply normalization and update the dataframe
boston_data[columns_after_normalization] = scaler.fit_transform(boston_data[columns_after_normalization])

# Display the first few rows to verify normalization
print(boston_data.head(100))
```

- Splits the standardized data into training and testing sets using train_test_split from scikit-learn.

```python
[156] # Train-test split
      X = boston_data.iloc[:, :-1].values
      y = boston_data.iloc[:, -1].values
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


      print("Training set size:", X_train.shape)
      print("Testing set size:", X_test.shape)

      Training set size: (404, 13)
      Testing set size: (102, 13)
```



.

- Reshapes y_train and y_test to ensure compatibility with the model.

- Prints the shapes of X_train, X_test, y_train, and y_test to inspect the dimensions of the data splits.
- Implemented Gradient Descent

```
[159] # Stochastic Gradient Descent Implementation
      class Stoc_Grad_Des:
          def __init__(self, learning_rate, epochs):
              self.m = None
              self.b = None
              self.lrn_rate = learning_rate
              self.epochs = epochs

          def fit(self, X_train, y_train):
              self.b = 0
              self.m = np.ones(X_train.shape[1])
              for _ in range(self.epochs):
                  for i in range(X_train.shape[0]):
                      y_hat = np.dot(X_train[i], self.m) + self.b
                      self.b -= self.lrn_rate * (-2 * (y_train[i] - y_hat))
                      self.m -= self.lrn_rate * (-2 * (y_train[i] - y_hat) * X_train[i])

          def predict(self, X_test):
              return np.dot(X_test, self.m) + self.b
```

- On the test split, computed mean square error and absolute error for evaluation.

```
[167] from sklearn.metrics import r2_score
      from sklearn import metrics
      print("MSE:",metrics.mean_squared_error(y_train,y_train_pred))
      print("MAE:",metrics.mean_absolute_error(y_train,y_train_pred))
      print("RMSE:",np.sqrt(metrics.mean_squared_error(y_train,y_train_pred)))
      print("R_squared:",r2_score(y_train,y_train_pred))

      MSE: 0.01130352473695203
      MAE: 0.0768031555136931
      RMSE: 0.10631803580273683
      R_squared: 0.7365173162048462
```

Task-4: Evaluation

```
[168] print("MSE:",metrics.mean_squared_error(y_test,y_test_pred))
      print("MAE:",metrics.mean_absolute_error(y_test,y_test_pred))
      print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test,y_test_pred)))
      print("R_squared:",r2_score(y_test,y_test_pred))

      MSE: 0.012414484577084514
      MAE: 0.07480547859538986
      RMSE: 0.11142030594592942
      R_squared: 0.6571930400258184
```

Conclusion:
In this report, we have successfully implemented various tasks related to Decision Trees and Linear Regression. These tasks involved data preprocessing, visualization, model implementation, evaluation, and performance analysis. By completing these tasks, we gained valuable insights into pattern recognition and machine learning techniques.

References:

https://github.com/datasciencedojo/datasets/blob/master/titanic.csv
https://raw.githubusercontent.com/devzohaib/Simple-Linear-Regression/master/tvmarketing.csv
https://lib.stat.cmu.edu/datasets/boston
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html#pandas.DataFrame.fillna
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
https://scikit-learn.org/stable/modules/model_evaluation.html
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html#pandas.DataFrame.drop
https://www.geeksforgeeks.org/ml-binning-or-discretization/
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.setp.html#matplotlib.pyplot.setp
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.suptitle.html#matplotlib.pyplot.suptitle
https://matplotlib.org/stable/gallery/shapes_and_collections/scatter.html#scatter-plot