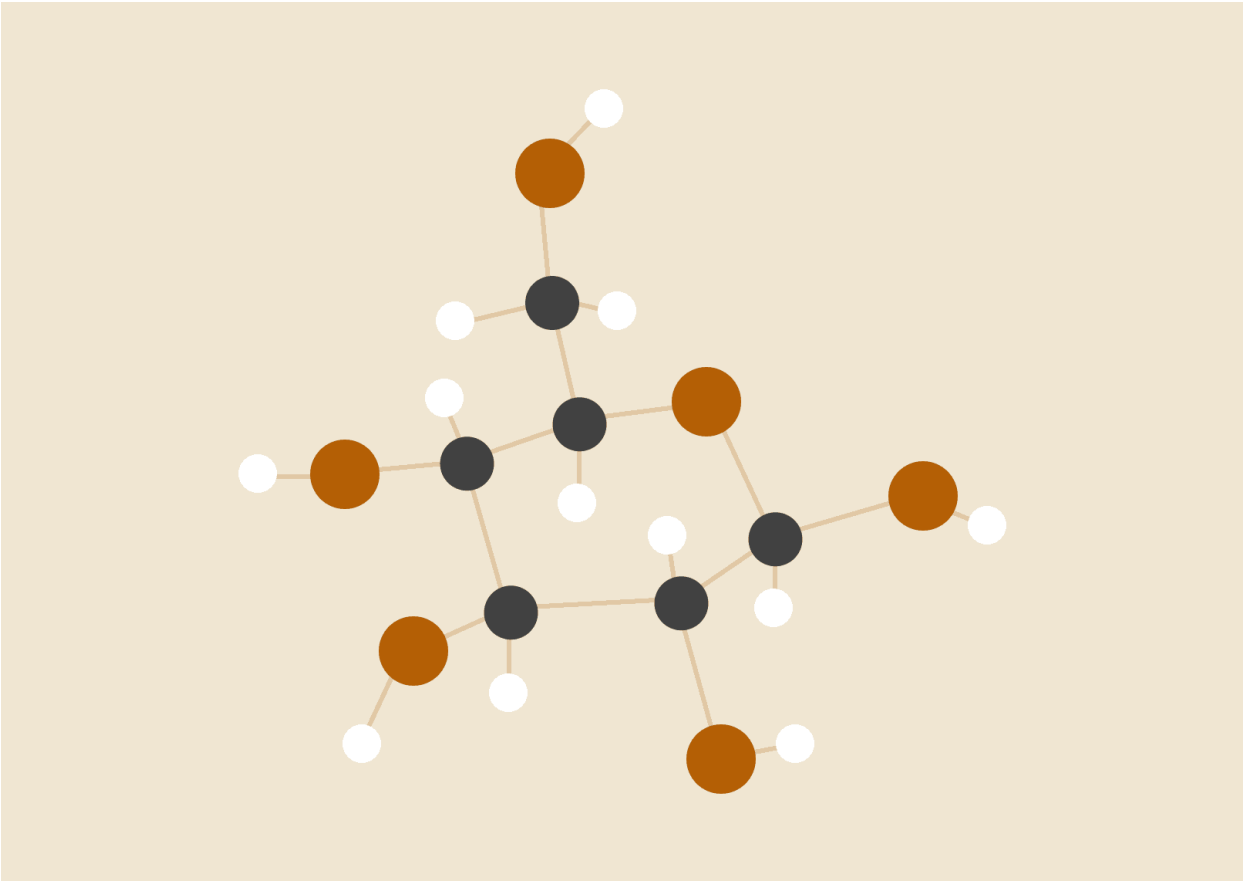
PRML Lab 2

Decision Tree | Linear Regression



Rhythm Patni

B22CS043

Question 1:

You are given a dataset. It contains data to classify whether someone will survive in the Titanic wreck. You need to implement a classification decision tree (DT) from scratch (you are not permitted to use any 3rd-party library's function for the classifier, e.g., Scikit. You may, however, use built-in functions for auxiliary tasks like train/test split, etc.).

Task 1

Perform pre-processing and visualization of the dataset. Also, men- tion in your report whether the given features are ordinal, nominal or cate- gorical. Perform categorical encoding wherever applicable and split the data into train, validation and test. Use a 70-20-10 split. (5 points)

The Raw Dataset is as follows:

```
df = pd.read_csv("titanic.csv")
df.head()
```

	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

The Analytics of the Data are as follows:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

The Data Contains **Missing Values** in Some Columns.

```
df.isna().sum()/df.shape[0] * 100
```

```
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
```

In the Age Column, Approximately 20% of the data is missing. Majority of the data is missing in the Cabin Column. A very small fraction of data is missing in the Embarked Column.

Age is a significant factor concerning one's survival. So we cannot simply drop 20% of the data simply because we don't have data. We will fill the NaN values in the Age column with the Average Age of a person in the same category in the entire dataset.

A Cabin Allocated to a person cannot affect his chances of Survival. So simply dropping the Cabin Column is no harm.

In the Embarked Column, we have exactly 2 NaN values. Since it is an insignificant portion of the data, we simply drop all the NaN values.

```
df.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
```

Now After Removing the NaN values, We remove the **Redundant Columns**.

Passenger Id : This is just an indexing given to the dataset. We simply drop it.

Name : Name of a person is merely an identifier. Inbuilt indexing in numpy and pandas can be used as an identifier of the person. So we also dropped the Name Column due to Redundancy.

Ticket : There is no significant piece of information in the Ticket ID of a person. The Data is irrelevant. Hence Dropped.

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

Classification of Columns:

The Columns in the Data are Classified as follows.

Pclass (Priority Class) : 'Ordinal' Data since we establish an ordering among first class,second class and third class passengers.

Sex : Sex of a person can be either 'Male' and 'Female', and no ordering can be established among these 2 sub-types, 'Sex' is 'nominal' Data.

Age : Age of a person can have infinitely many values, and they can be ordered based on magnitude. So Age is 'Continuous Data'.

SibSp (Sibling and Spouse) : The Number of Siblings and Spouses of a Person can have

Integer values only and an order can be established, SibSp is ‘Ordinal’ Data.

Parch (Parent and Children) : Similar to SibSp, Parch is also ‘Ordinal’ Data

Fare : Similar to ‘Age’, ‘Fare’ is also ‘Continuous’ Data

Embarked : The Place from where a person Embarked there Journey on the Titanic can only have Discrete values and no ordering can be established hence, ‘Embarked’ is ‘Ordinal’ Data.

Categorical Encoding:

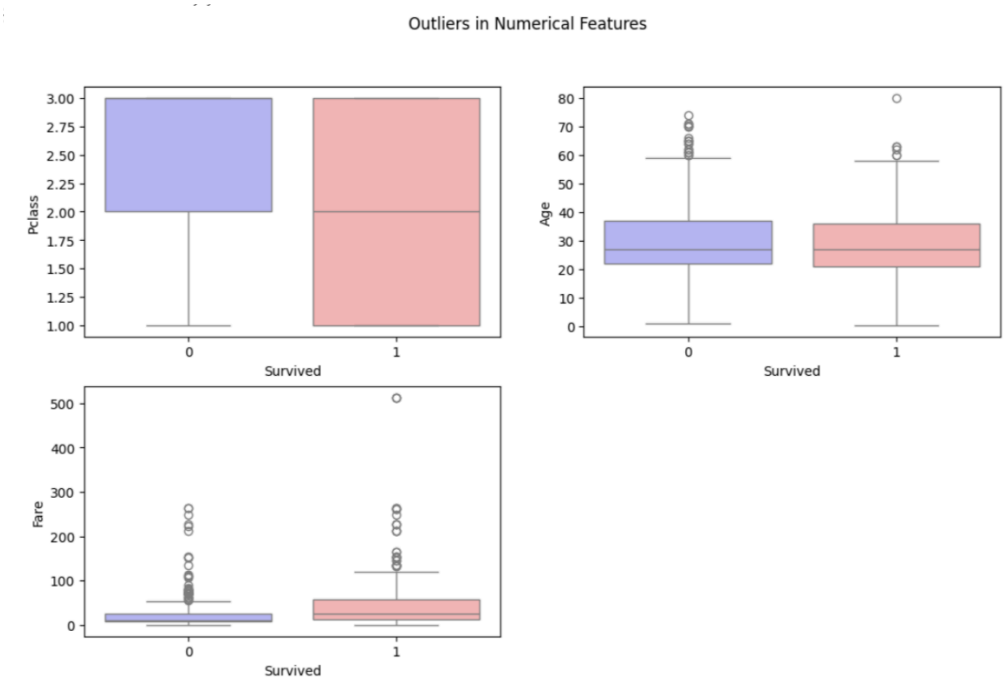
For this Data we Encode as Follows:

For the Sex Column, ‘male’ is encoded as 1 and ‘female’ is encoded as ‘0’

For the Embarked Column, ‘S’ is encoded is 0, ‘C’ is encoded as ‘1’ , ‘Q’ is encoded as ‘2’

Final Dataset we can work with:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Survived
0	3	1	22.0	1	0	7.2500	0	0
1	1	0	38.0	1	0	71.2833	1	1
2	3	0	26.0	0	0	7.9250	0	1
3	1	0	35.0	1	0	53.1000	0	1
4	3	1	35.0	0	0	8.0500	0	0



The analysis of these plots leads to the following observations:

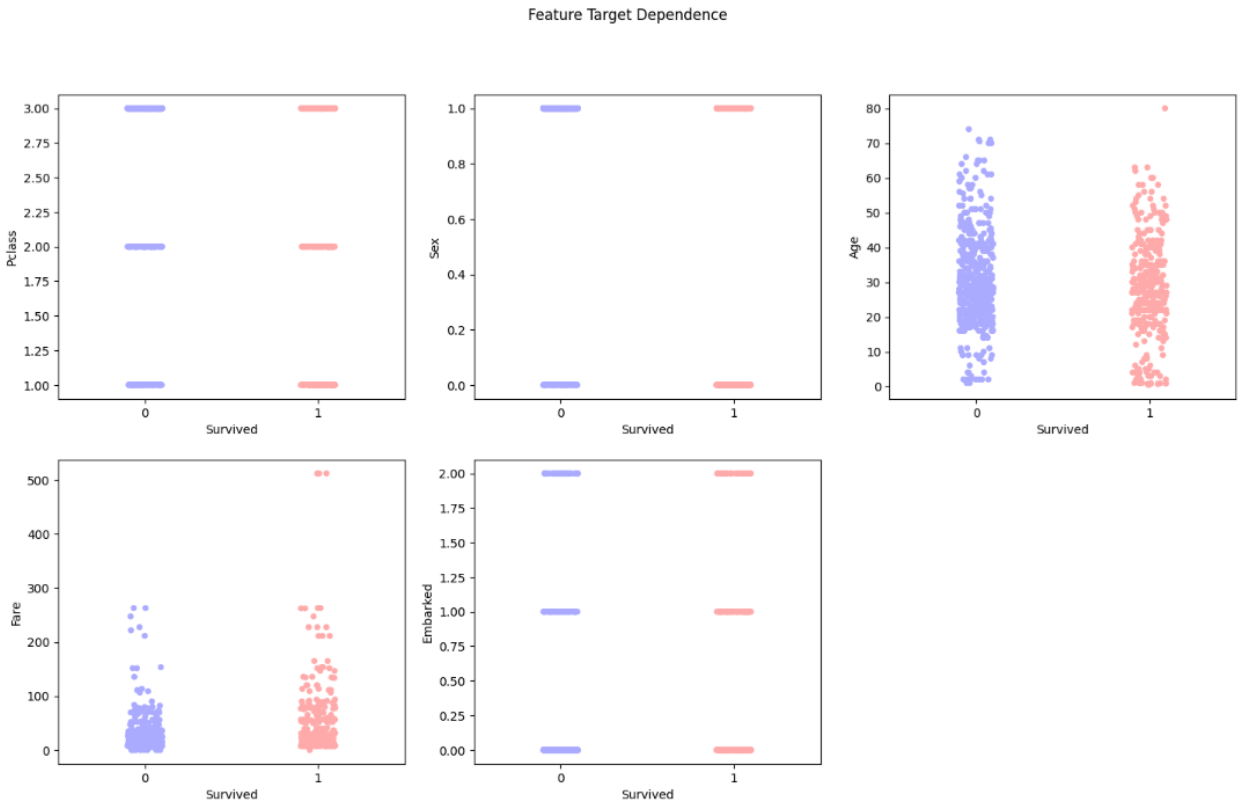
Pclass: This particular feature encompasses only three discrete values.

Age: The plots reveal data points exceeding 60, which signifies the presence of elderly individuals on the ship rather than outliers. Thus, it is deemed relevant.

Fare: The plot displays substantial fare values, which is deemed reasonable. Therefore, it is also taken into consideration.

Pclass: With only three discrete values (1, 2, and 3), no outliers are identified.

In summary, based on the above considerations, it can be concluded that there are no significant outliers present in the given dataset.



Task 2

Implement the entropy as the cost function to calculate the split.

```
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
```

The Function has been implemented inside the DecisionTreeClassifier Class.

Task 3,4,5

conToCat() Functionality is implemented in the split function of the class, the Splits are made after checking the columns of the split first.

For The Implementation of the Decision Tree, Two Classes are implemented , The Node Class and the DecisionTreeClassifier Class.

Infer() Function is implemented inside the DecisionTreeClassifier Class, To make predictions on a single Data Point. This Function will be used to find the Accuracy of the model.

Task 6,7,8

Function to find Accuracy, Precision,Recall,Confusion Matrix, and F1 Score have been implemented.

Using these functions, The final analysis of the model is as follows.

```
classifier = DecisionTreeClassifier(min_samples_split=3, max_depth=5)
classifier.fit(X_train,Y_train)
Y_pred_test = classifier.predict(X_test)
Y_pred_train = classifier.predict(X_train)
Y_pred_val = classifier.predict(X_val)
Y_val_normalized = Y_val.ravel()
Y_test_normalized = Y_test.ravel()
Y_train_normalized = Y_train.ravel()
print("Accuracy on the Testing data is",AccuracyScore(Y_test_normalized, Y_pred_test) * 100)
print("Accuracy on the Training data is",AccuracyScore(Y_train_normalized, Y_pred_train) * 100)
print("Accuracy on the Validation data is",AccuracyScore(Y_val_normalized, Y_pred_val) * 100)
```

```
Accuracy on the Testing data is 82.02247191011236
Accuracy on the Training data is 86.49517684887459
Accuracy on the Validation data is 79.7752808988764
```

```
print("Confusion Matrix: ", confusion_matrix(Y_test_normalized, Y_pred_test))
print("Precision of 'Died' Class:",precision_per_class(Y_test_normalized,Y_pred_test,class_label = 0))
print("Precision of 'Alive' Class:",precision_per_class(Y_test_normalized,Y_pred_test,class_label = 1))
print("Recall of 'Died' Class:",recall_per_class(Y_test_normalized,Y_pred_test,class_label = 0))
print("Recall of 'Alive' Class:",recall_per_class(Y_test_normalized,Y_pred_test,class_label = 1))
print("f1-score of 'Died' Class:",f1_score_per_class(Y_test_normalized,Y_pred_test,class_label = 0))
print("f1-score of 'Alive' Class:",f1_score_per_class(Y_test_normalized,Y_pred_test,class_label = 1))
print("Support of 'Died' Class:",support_per_class(Y_test_normalized,class_label = 0))
print("Support of 'Alive' Class:",support_per_class(Y_test_normalized,class_label = 1))
print("Accuracy of 'Died' Class,",classwise_accuracy(Y_test_normalized,Y_pred_test,class_label = 0)*100)
print("Accuracy of 'Alive' Class,",classwise_accuracy(Y_test_normalized,Y_pred_test,class_label = 1)*100)
```

```
Confusion Matrix:  [[22, 4], [12, 51]]
Precision of 'Died' Class: 0.81
Precision of 'Alive' Class: 0.85
Recall of 'Died' Class: 0.93
Recall of 'Alive' Class: 0.65
f1-score of 'Died' Class: 0.87
f1-score of 'Alive' Class: 0.74
Support of 'Died' Class: 55
Support of 'Alive' Class: 34
Accuracy of 'Died' Class, 92.72727272727272
Accuracy of 'Alive' Class, 64.70588235294117
```

Accuracy on the Testing data is 82.02% and on the Training data is 86.49%.All other analytics are displayed as above.

Question 2:

Given a dataset with two features: TV marketing budget and sales,the following tasks are performed.

Task 1

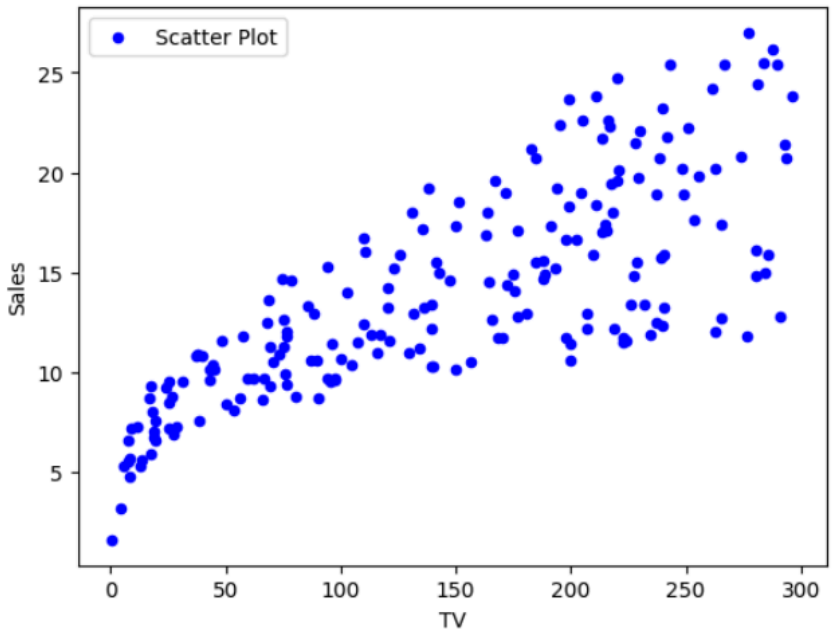
Dataset Exploration: (a) Load the dataset and display the first few rows. (b) Plot a scatter plot to visualize the relationship between the TV marketing budget and sales. Comment on the trend observed in the scatter plot. (c) Calculate and display basic statistical measures (mean, standard deviation) for both TV marketing budget and sales.

The Plot of the raw data is as follows.

```
# Load the dataset
dataset_url = "https://raw.githubusercontent.com/devzohaib/Simple-Linear-Regression/master/tvmarketing.csv"
df = pd.read_csv(dataset_url)
```

```
df.plot.scatter(x='TV', y='Sales', marker='o', color='blue', label='Scatter Plot')
```

```
<Axes: xlabel='TV', ylabel='Sales'>
```



The Head of the Data is

	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 Statistics of the Data are as follows.

	TV	Sales
count	200.000000	200.000000
mean	147.042500	14.022500
std	85.854236	5.217457
min	0.700000	1.600000
25%	74.375000	10.375000
50%	149.750000	12.900000
75%	218.825000	17.400000
max	296.400000	27.000000

Task 2

Checking for missing data…….

```
df.isna().sum()
```

```
TV       0
Sales    0
dtype: int64
```

As we can see above, there is no missing data in the dataset, So we will not drop anything.

Normalization of the data is done using Z-Score Normalization.

```
def z_score_normalization(data):
    # Calculate mean and standard deviation for each column
    means = data['TV'].mean()
    stds = data['TV'].std()

    # Z Score Normalization
    data['TV'] = (data['TV'] - means) / stds

    return data
```

```
df = z_score_normalization(df)
df.head()
```

	TV	Sales
0	0.967425	22.1
1	-1.194379	10.4
2	-1.512360	9.3
3	0.051919	18.5
4	0.393196	12.9

Custom Train-Test-Split function is implemented to perform split.

Task 3

GetBestSplit() Function is implemented to perform gradient descent on the dataset.

MeanSquaredError() and MeanAbsoluteError() functions are implemented to test the model.

Task 4

```
[1867] def MeanSquaredError(y_test,y_pred):
         return np.mean((y_test - y_pred)**2)

       def MeanAbsoluteError(y_test,y_pred):
         return np.mean((np.abs(y_test - y_pred)))
```

```
y_test = test_df['Sales']
x_test = test_df['TV']
y_pred = slope*x_test + intercept
mse = MeanSquaredError(y_test,y_pred)
mae = MeanAbsoluteError(y_test,y_pred)
print("Mean Squared Error =",mse)
print("Mean Absolute Error =",mae)
```

```
Mean Squared Error = 13.660771747919403
Mean Absolute Error = 2.933078434764753
```

The Mean Squared Error is coming out to be 13.66 and Mean Absolute Error is coming out to be 2.93.

Question 3:

Repeating all the tasks in Problem-2 for predicting house rent on the Boston Housing dataset. Note that in this dataset, you need to perform multivariate linear regression.

Task 1

```
data = pd.read_csv("http://lib.stat.cmu.edu/datasets/boston", sep="\s+", skiprows=22,header=None)
```

```
X3 = np.hstack([data.values[::2, :], data.values[1::2, :2]])
y3 = data.values[1::2,2]
```

```
df3 = pd.DataFrame({'CRIM':X3[:,0], 'ZN':X3[:,1], 'INDUS':X3[:,2], 'CHAS':X3[:,3], 'NOX':X3[:,4], 'RM'
df3.head()
```

	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	1.0	296.0	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	2.0	242.0	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	2.0	242.0	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	3.0	222.0	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	3.0	222.0	18.7	396.90	5.33	36.2

The Statistics of the code are as follows

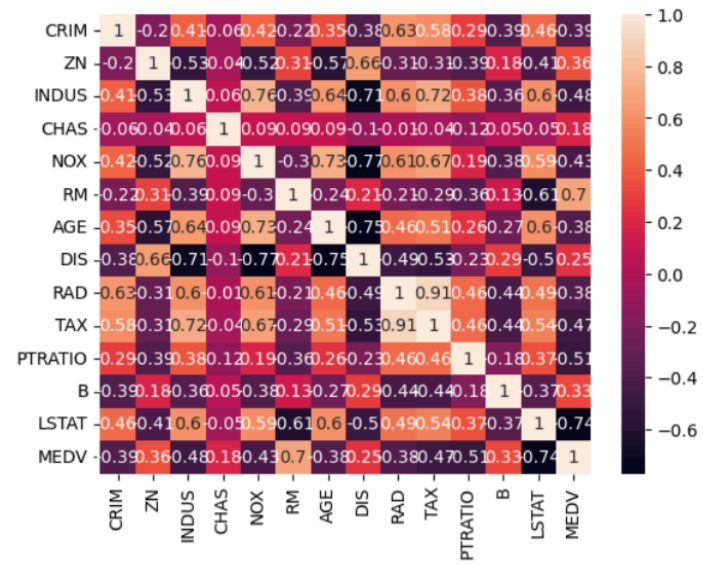
```
df3.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.653063	22.532806
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.141062	9.197104
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.730000	5.000000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.950000	17.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.360000	21.200000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.955000	25.000000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.970000	50.000000

The Heatmap of the code is plotted as shown:

```
# sns.heatmap(df3)
correlation_matrix = df3.corr().round(2)
sns.heatmap(data = correlation_matrix,annot = True)
```

```
<Axes: >
```



Task 2

Checking for NaN Values.

```
df3.isna().sum()
```

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

We can see that there are no NaN values. So no data will be deleted.

Task 3

Multiple Linear Regression is implemented using the class MultipleLinearRegression. Gradient Descent is used to get the Best Fit.

Task 4

The Model is tested using Mean Squared Error and Mean Absolute Errors as shown.

Task 4 : Evaluating the Model.

```
Regressor = MultipleLinearRegressor()
Regressor.fit(X_train,Y_train)
y_pred = Regressor.predict(X_test)
# type(Y_test)
Y_test_normal = np.array(list(Y_test))
mse = MeanSquaredError(Y_test_normal,y_pred)
mae = MeanAbsoluteError(Y_test_normal,y_pred)
print("Mean Squared Error =",mse)
print("Mean Absolute Error =",mae)
```

```
Mean Squared Error = 23.59084704575107
Mean Absolute Error = 3.473453799113906
```