```
import pandas as pd
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix,r2_score,mean_squared_error,precision_recall_fscore_support
from sklearn import preprocessing
import matplotlib.pyplot as plt
import seaborn as sns
```

1.

```
df = pd.read_csv('https://raw.githubusercontent.com/Deepsphere-AI/DataAnalyticsTraining/main/PredictiveAnalytics/housing.csv')
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	med
0	-122.23	37.88	41	880	129.0	322	126	8.3252	NEAR BAY	
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	NEAR BAY	
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	NEAR BAY	
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	NEAR BAY	
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	NEAR BAY	

```
df.shape
```

```
(20640, 10)
```

```
#1a
df.isna().sum() # Total_bedrooms has null values
```

```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
ocean_proximity         0
median_house_value      0
dtype: int64
```

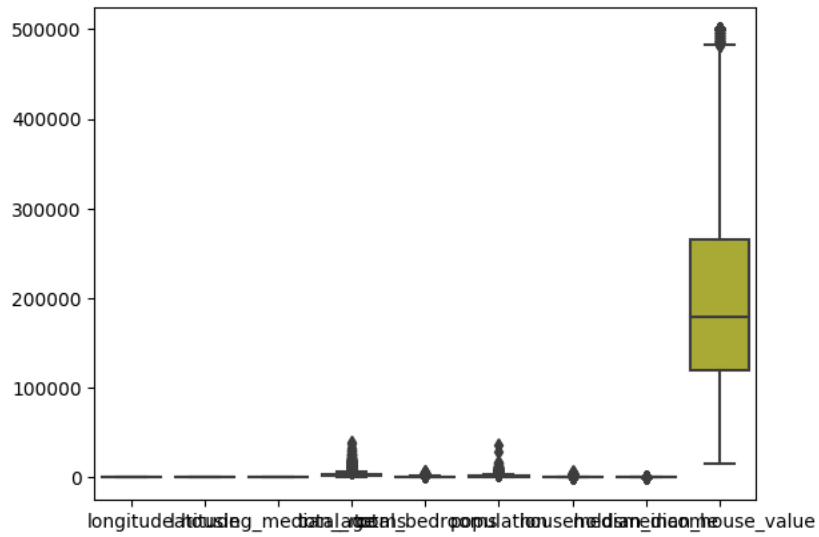
```
df =df.fillna(method='ffill')
```

```
df.isna().sum() # Null values handled.
```

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
ocean_proximity       0
median_house_value    0
dtype: int64
```

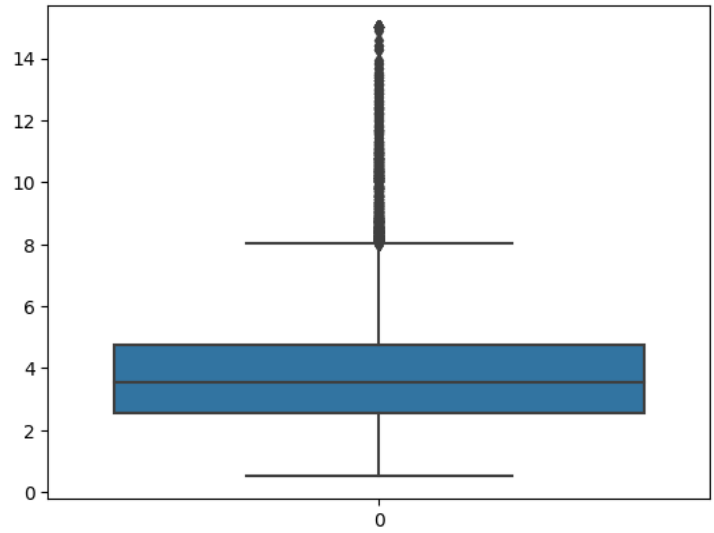
```
sns.boxplot(df)
```

```
<Axes: >
```




```
sns.boxplot(df['median_income'])
```

```
<Axes: >
```




```
df['ocean_proximity'].unique()
```

```
array(['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND'],
      dtype=object)
```

```
#Data Preprocessing

#1b
label_encoder = preprocessing.LabelEncoder()
df['ocean_proximity'] = label_encoder.fit_transform(df['ocean_proximity'])
df['ocean_proximity'].unique()
```

```
array([3, 0, 1, 4, 2])
```

```
#1c
print(df.corr(),'\n')
print("Number of duplicates : ",df.duplicated().sum())
#No Duplicate values present in the data
```

```
                    longitude  latitude  housing_median_age  total_rooms  \
longitude            1.000000 -0.924664           -0.108197     0.044568
latitude            -0.924664  1.000000            0.011173    -0.036100
housing_median_age  -0.108197  0.011173            1.000000    -0.361262
total_rooms          0.044568 -0.036100           -0.361262     1.000000
total_bedrooms       0.070442 -0.067535           -0.319312     0.925347
population           0.099773 -0.108785           -0.296244     0.857126
households           0.055310 -0.071035           -0.302916     0.918484
median_income       -0.015176 -0.079809           -0.119034     0.198050
ocean_proximity     -0.289779  0.200974            0.112468    -0.015693
median_house_value  -0.045967 -0.144160            0.105623     0.134153
```

```
                        total_bedrooms  population  households  median_income  \
longitude                     0.070442    0.099773    0.055310      -0.015176
latitude                     -0.067535   -0.108785   -0.071035      -0.079809
housing_median_age           -0.319312   -0.296244   -0.302916      -0.119034
total_rooms                   0.925347    0.857126    0.918484       0.198050
total_bedrooms                1.000000    0.872491    0.972942      -0.007473
population                    0.872491    1.000000    0.907222       0.004834
households                    0.972942    0.907222    1.000000       0.013033
median_income                -0.007473    0.004834    0.013033       1.000000
ocean_proximity              -0.014702   -0.070282   -0.018186      -0.014957
median_house_value            0.049294   -0.024650    0.065843       0.688075

                        ocean_proximity  median_house_value
longitude                     -0.289779           -0.045967
latitude                       0.200974           -0.144160
housing_median_age             0.112468            0.105623
total_rooms                   -0.015693            0.134153
total_bedrooms                -0.014702            0.049294
population                    -0.070282           -0.024650
households                    -0.018186            0.065843
median_income                 -0.014957            0.688075
ocean_proximity                1.000000            0.081750
median_house_value             0.081750            1.000000

Number of duplicates :  0
```

```
#Splitting the dataset into train and test data
#1c
X = df.iloc[:,:9]
y = df.iloc[:,9:]
print("Columns in X : \t ", X.columns,"\n\n")
print("Columns in y : \t",y.columns)
```

```
Columns in X :    Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'ocean_proximity'],
      dtype='object')


Columns in y :   Index(['median_house_value'], dtype='object')
```

```
#1d
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state = 42 , test_size= 0.2)
```

```
#1e
#Linear Regression Model
reg =  LinearRegression()
reg.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
y_pred = reg.predict(X_test)
y_pred
```

```
array([[ 75631.25398983],
       [198325.61748579],
       [299567.35453823],
       ...,
       [439690.11277371],
       [130408.05181885],
       [175513.56755483]])
```

```
#1f
R_Squared = r2_score(y_test,y_pred)*100
print("R_Squared : \n",R_Squared)
RMSE   = mean_squared_error(y_test,y_pred)
print(" RMSE : \n",RMSE)
```

```
R_Squared :
 61.14554518898516
 RMSE :
 5091522642.900918
```

2.

```
#2
df1 = pd.read_csv('https://raw.githubusercontent.com/Deepsphere-AI/DataAnalyticsTraining/main/PredictiveAnalytics/Iris.csv')
df1.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
df1.shape
```

```
(150, 6)
```

```
df1.dtypes
```

```
Id                int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

```
#2a
df1.isna().sum() #No Missing Values present in the dataset.
```

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

```
df1['Species'].unique()
```

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

```
#2b
#Data Encoding
label_encoder = preprocessing.LabelEncoder()
df1['Species'] = label_encoder.fit_transform(df1['Species'])
df1['Species'].unique()
```

```
array([0, 1, 2])
```

```
#2c
df1.duplicated() #No duplicate records present in the dataset
```

```
0      False
1      False
2      False
3      False
4      False
       ...
145    False
146    False
147    False
148    False
149    False
Length: 150, dtype: bool
```

```
print(df1.corr())
```

```
                     Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  \
Id             1.000000       0.716676     -0.397729       0.882747
SepalLengthCm  0.716676       1.000000     -0.109369       0.871754
SepalWidthCm  -0.397729      -0.109369      1.000000      -0.420516
PetalLengthCm  0.882747       0.871754     -0.420516       1.000000
PetalWidthCm   0.899759       0.817954     -0.356544       0.962757
Species        0.942830       0.782561     -0.419446       0.949043

               PetalWidthCm   Species
Id                 0.899759  0.942830
```

```
SepalLengthCm      0.817954  0.782561
SepalWidthCm      -0.356544 -0.419446
PetalLengthCm      0.962757  0.949043
PetalWidthCm       1.000000  0.956464
Species            0.956464  1.000000
```

```
X =df1.iloc[:,:5]
y = df1.iloc[:,5]
```

```
#2d
#Splitting the source dataset into training and testing data
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state = 0, test_size= 0.2)
```

```
#2e
clf = tree.DecisionTreeClassifier()
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
```

```
#2f
accuracy = accuracy_score(y_test,y_pred)*100
print("accuracy : ",accuracy)
print('Precision ,Recall, F1Score ',precision_recall_fscore_support(y_test,y_pred))
```

```
accuracy :  96.66666666666667
Precision ,Recall, F1Score  (array([1.        , 0.92857143, 1.        ]), array([1.        , 1.        , 0.83333333]), array([1.
```