2 Programming

2.1 Decision Tree

In this question, we implement Decision Tree, Bagging of Trees and Random Forests to predict the sale of the Carseats dataset. All the algorithms can be implemented by sklearn, the loss will be set as MSE.

2.1.1 Data Statistics

There are 11 columns of the dataset (including sales), 8 of them are numeric type, 3 of them are string type ('ShelveLoc', 'Urban', 'US').

The details of the numerical data are as follows:

	Sales	CompPrice	Income	Advertising	Population	Price	Age	Education
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	7.496325	124.975000	68.657500	6.635000	264.840000	115.795000	53.322500	13.900000
std	2.824115	15.334512	27.986037	6.650364	147.376436	23.676664	16.200297	2.620528
min	0.000000	77.000000	21.000000	0.000000	10.000000	24.000000	25.000000	10.000000
25%	5.390000	115.000000	42.750000	0.000000	139.000000	100.000000	39.750000	12.000000
50%	7.490000	125.000000	69.000000	5.000000	272.000000	117.000000	54.500000	14.000000
75%	9.320000	135.000000	91.000000	12.000000	398.500000	131.000000	66.000000	16.000000
max	16.270000	175.000000	120.000000	29.000000	509.000000	191.000000	80.000000	18.000000

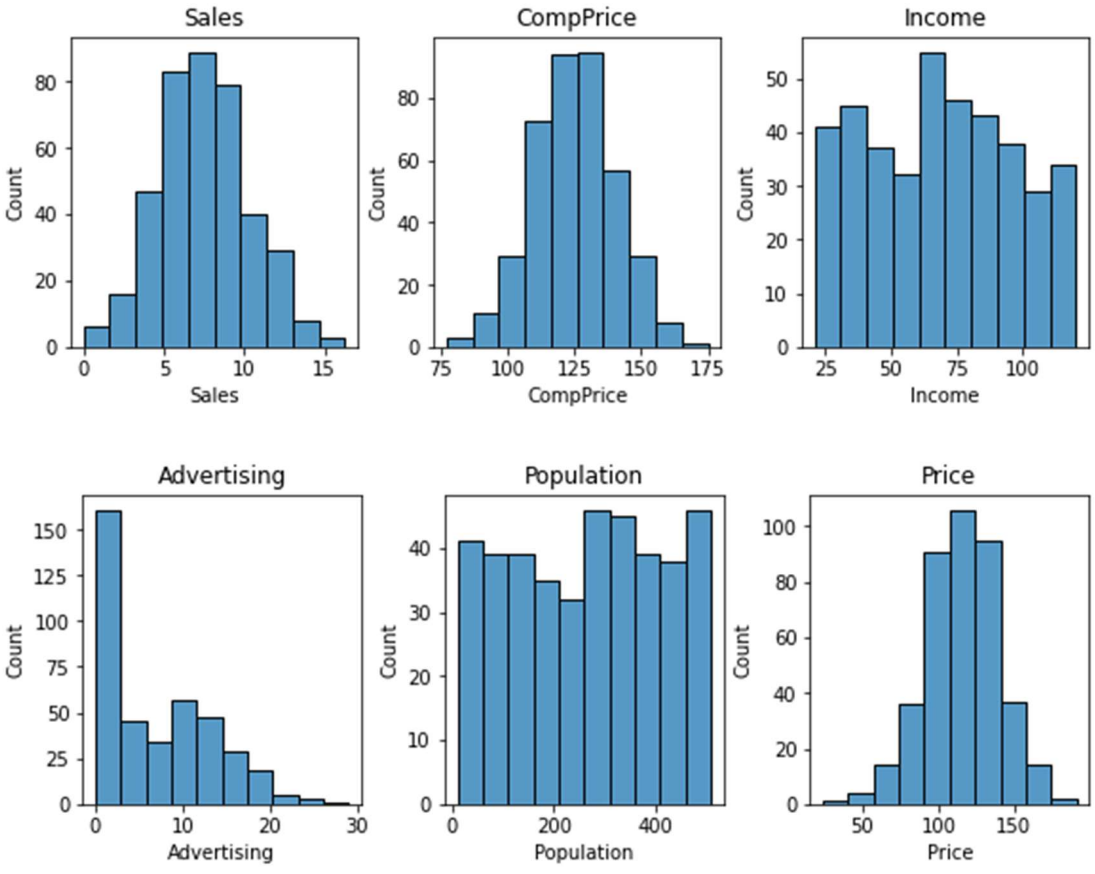
The details of the categorical data are as follows:

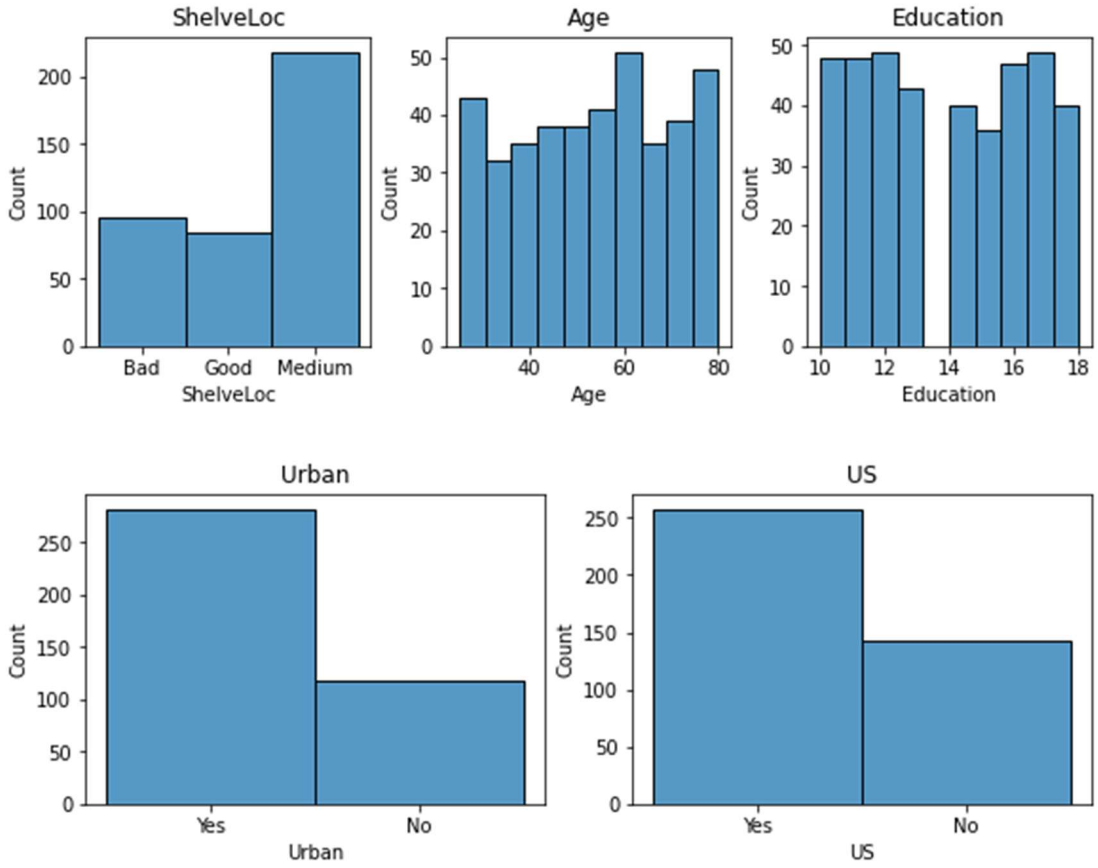
```
Medium    219
Bad        96
Good       85
Name: ShelveLoc, dtype: int64
```

```
Yes    282
No     118
Name: Urban, dtype: int64
```

```
Yes    258
No     142
Name: US, dtype: int64
```

Also, I draw the histograms of each column, and use seaborn to visualize them.





Data Preprocessing:

1. Since there're 3 categorical variables, we cannot use it directly to do regressions. It's straightforward to think about one-hot encoding in this scenario since each of the three categorical variables has less than or equal to 3 categories. Thus, I use one-hot encoding to change the variable. There's also no need to do normalization since decision tree is not sensitive to the scale of the numerical data.
2. For the training and testing data, I choose the first option, that is, I use the first 300 rows as the training set, and the remaining 100 rows as the testing set.

2.1.2 Decision Tree

First, I use GridSearchCV offered by sklearn to search for the best maximum depth and least node size. The result is as follows:

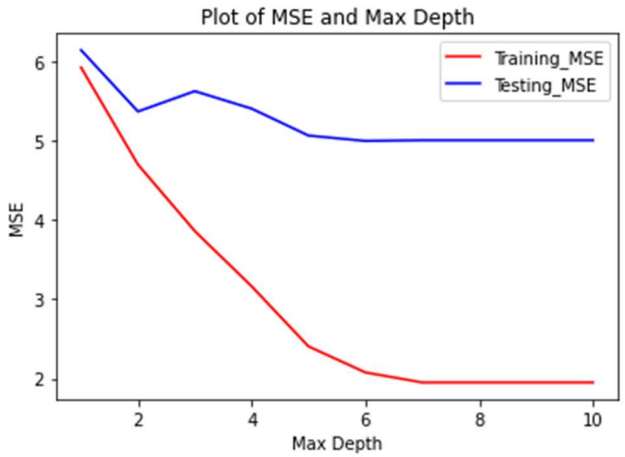
```
Best Parameter Pairs:  {'max_depth': 8, 'min_samples_leaf': 9}
```

Then, I define a function decision_tree (maxdepth, minleafsamples), which use DecisionTreeRegressor in sklearn to do the task and return its training and testing mse by metrics.mean_squared_error ().

Result Analysis:

1. First, I change maximum depth from 1 to 10, while maintaining least node sizes as 9, according to the GridSearchCV.

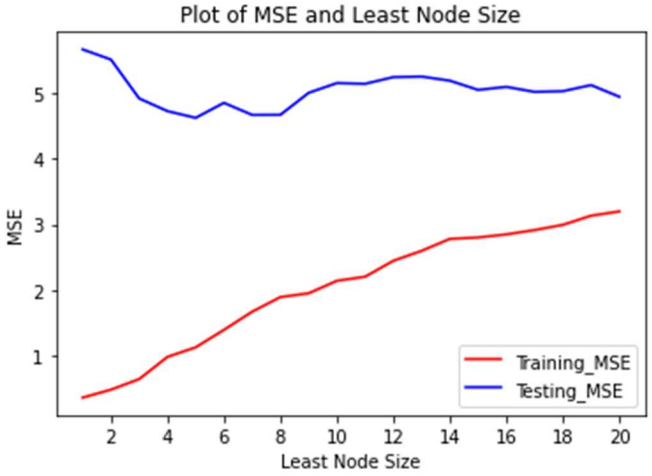
The result is shown below:



From this figure, we find that with the increase of max depth, which means that model complexity is increasing, the train error always decreases. The test error decrease at first and start be stable when the depth is around 5. It proves that when the complexity is too small, the model will be underfitted, both training and testing error is high. When the complexity is too large, it tends to be overfitted. The training error is smaller when increasing the max depth, but the testing error is stable.

2. Then I change least node size from 1 to 20, while maintaining maximum depth as 8, according to the GridSearchCV.

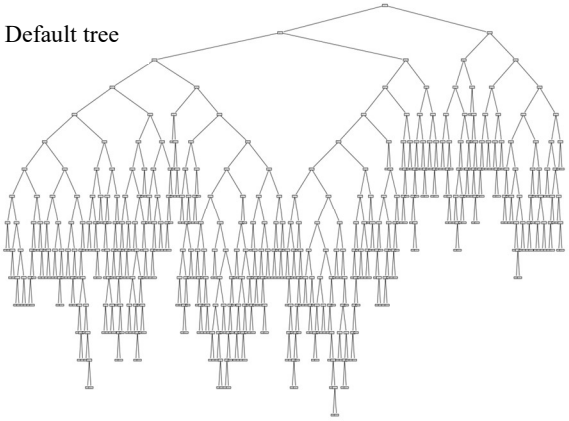
The result is shown below:



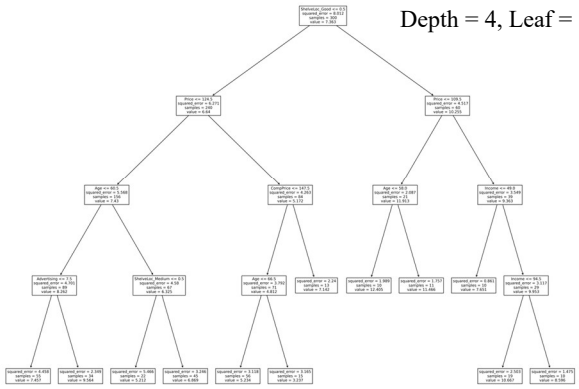
From this figure, we find that with the increase of node size, which means that model complexity is decreasing, the training error always increases. The testing error decreases first and then increases. This proves that when the node size is small, the model tends to be overfitted, when the node size is large, the model tends to be underfitted.

Below are some of the Decision Tree plots, each of which has different set of parameters. (If the plot shown below is hard to recognize, you can directly find them in the assignment folder, whose names are p1.png, p2.png, etc. They will be much clearer.)

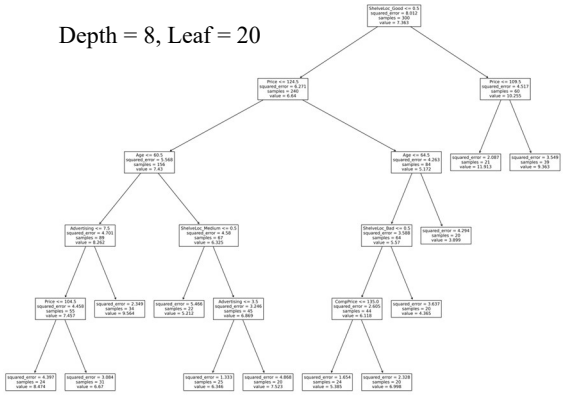
Default tree



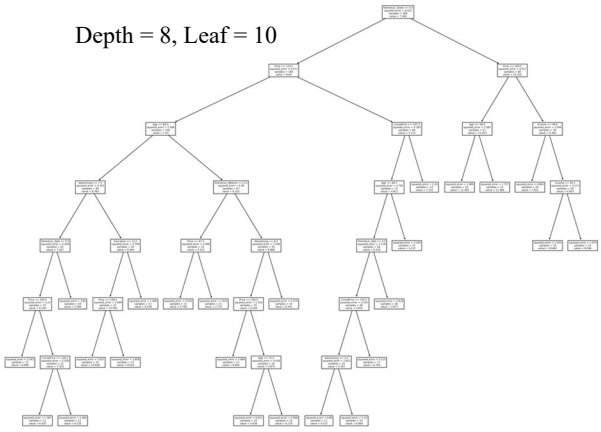
Depth = 4, Leaf = 10



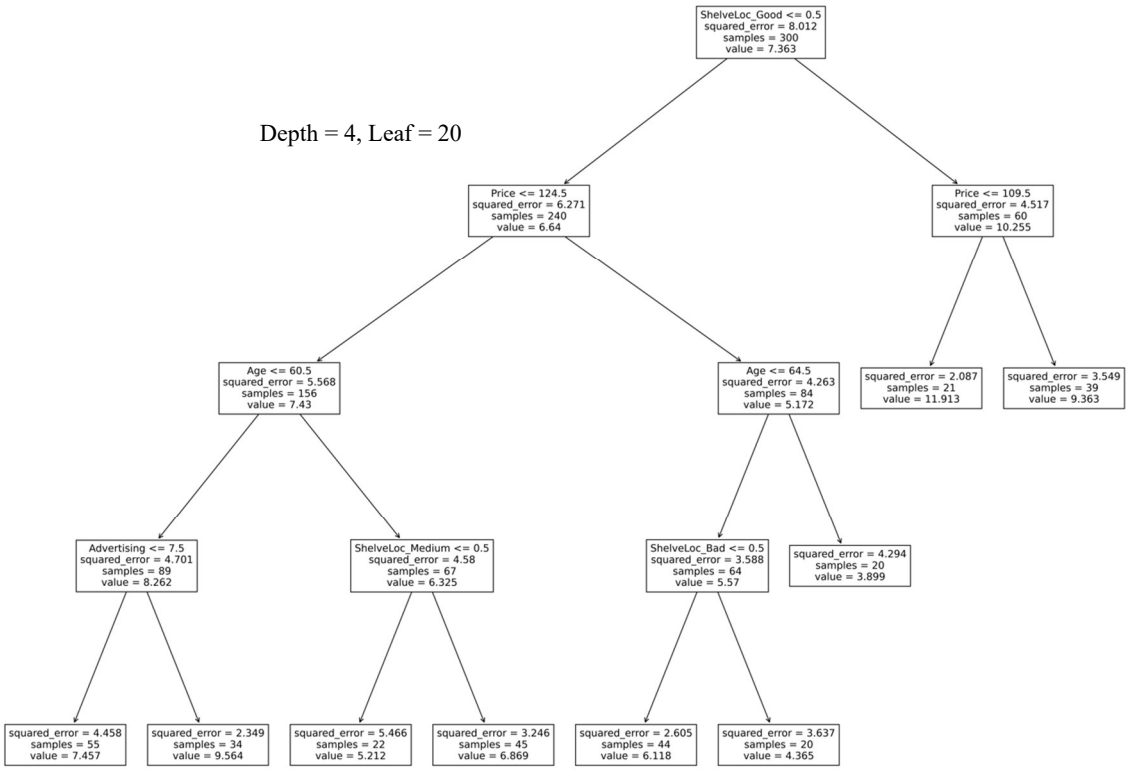
Depth = 8, Leaf = 20



Depth = 8, Leaf = 10



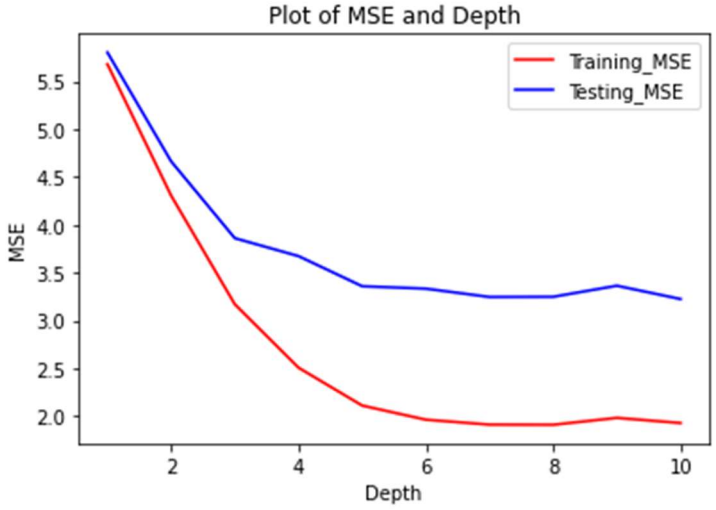
Depth = 4, Leaf = 20



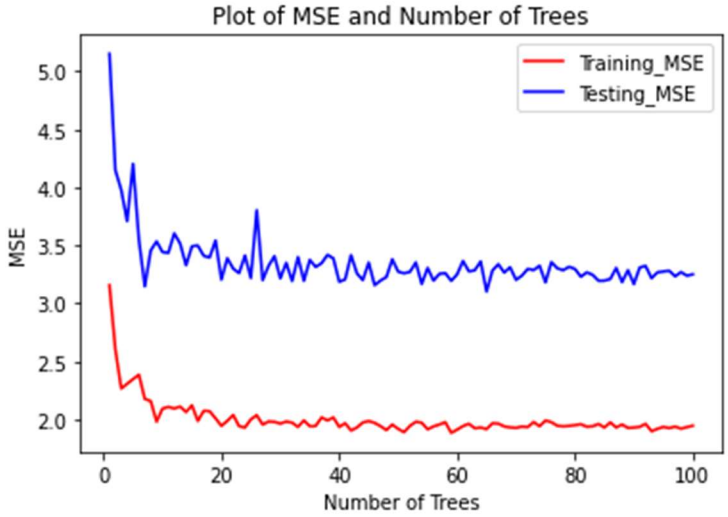
2.1.3 Bagging of Trees

First, I define a function bagging (depth, numtrees), which use BaggingRegressor in the outer space and DecisionTreeRegressor in the inner space from sklearn to do the task and

return its training and testing mse by metrics.mean_squared_error ().
Result Analysis:
1. First, I change maximum depth from 1 to 10, while maintaining the number of trees as 50. The result is shown below:



From this figure, we find that with the increase of depth, which means that model complexity is increasing, the training error always decreases. The testing error decreases at first and start to be stable when the depth is larger than 6. It proves that when the depth is too small, the model will be underfitted, when the depth is too large, the model might be overfitted. (E.g. Depth = 100)
2. Then, I change the number of trees from 1 to 100, while maintaining the maximum depth as 6. The result is shown below:



From this figure, we find that when the number of trees increase, the training error and testing error start to decrease at first and then be stable. The number of trees means data-level randomness, has no relationship with complexity since each tree may have roughly the same depth.

2.1.4 Random Forests

First, I use GridSearchCV offered by sklearn to search for the best number of trees, value of m, maximum depth and least node size. The result is as follows: