Homework4

20200639 Chae Woojin

Chapter 8: 2, 3, 4, 5, 8, 11

Chapter 11: 2, 3, 4, 7, 8, 11

Chapter 8:

#2.

A decision in which a single split occurs on the variable X_j(at value s) can be simplified as follows.

$$f(X) = c_1 \cdot I(X_j < s) + c_2 \cdot I(X_j \geq s)$$

With boosted stumps, following algorithm gives final decision as below.

b=1:

$$\hat{f}^1(X) = \alpha_1 \cdot I(X_{j_1} < s_1) + \beta_1 \cdot I(X_{j_1} \geq s_1)$$
$$\hat{f}(X) \leftarrow \hat{f}(X) + \lambda \hat{f}^1(X)$$
$$= \lambda \hat{f}^1(X)$$

$$r_i \leftarrow r_i - \lambda \hat{f}^1(X)$$
$$= y_i - \lambda \hat{f}^1(X)$$

b=2:

$$\hat{f}^2(X) = \alpha_2 \cdot I(X_{j_2} < s_2) + \beta_2 \cdot I(X_{j_2} \geq s_2)$$
$$\hat{f}(X) \leftarrow \hat{f}(X) + \lambda \hat{f}^2(X)$$
$$= \lambda \hat{f}^1(X) + \lambda \hat{f}^2(X)$$

$$r_i \leftarrow r_i - \lambda \hat{f}^2(X)$$
$$= y_i - \lambda \hat{f}^1(X) - \lambda \hat{f}^2(X)$$

⋮

b=B:

$$\hat{f}^B(X) = \alpha_B \cdot I(X_{j_B} < s_B) + \beta_B \cdot I(X_{j_B} \geq s_B)$$
$$\hat{f}(X) \leftarrow \hat{f}(X) + \lambda \hat{f}^B(X)$$
$$= \lambda \hat{f}^1(X) + \lambda \hat{f}^2(X) + \cdots + \hat{f}^B(X)$$

$$r_i \leftarrow r_i - \lambda \hat{f}^B(X)$$
$$= y_i - \lambda \hat{f}^1(X) - \lambda \hat{f}^2(X) - \cdots - \hat{f}^B(X)$$

After Bth iteration, one could get final boosting model $\hat{f}$, given by

$$\hat{f}(X) = \sum_{b=1}^{B} \lambda \hat{f}^b(X) = \lambda \sum_{b=1}^{B} \alpha_b \cdot I\left(X_{j_b} < s_b\right) + \beta_b \cdot I\left(X_{j_b} \geq s_b\right)$$

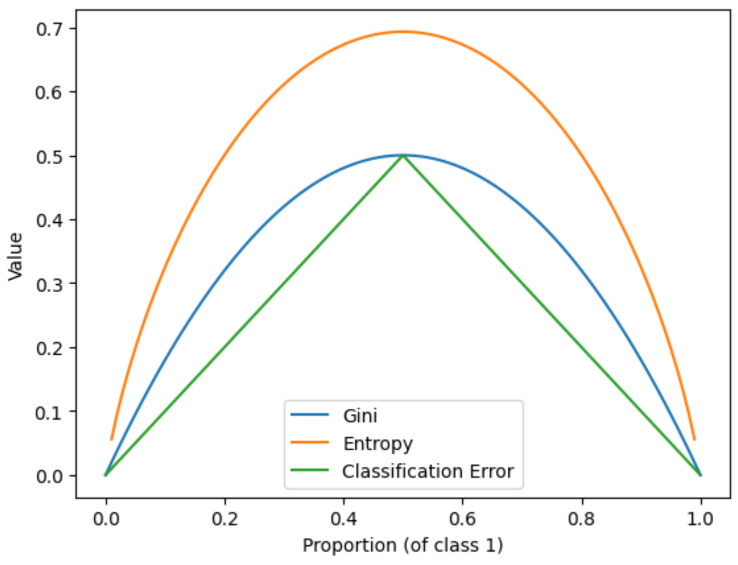
Since $X_{j_1}, X_{j_2}, \cdots, X_{j_B} \in \left\{X_1, X_2, \cdots, X_p\right\}$, we can see that this estimate $\hat{f}$ is indeed an additive model of the form $f(X) = \sum_{j=1}^{p} f_j(X_j)$.

#3.

Gini index is defined by $G = \sum_{k=1}^{2} \hat{p}_{mk}(1 - \hat{p}_{mk}) = \hat{p}_{m1}(1 - \hat{p}_{m1}) + \hat{p}_{m2}(1 - \hat{p}_{m2})$.

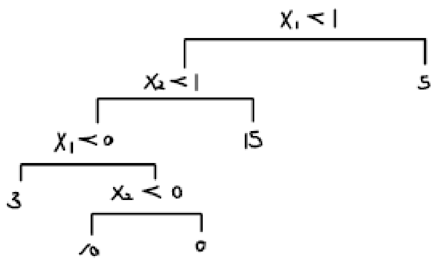
Entropy is defined by $D = -\sum_{k=1}^{2} \hat{p}_{mk} \log(\hat{p}_{mk}) = -\hat{p}_{m1} \log(\hat{p}_{m1}) - \hat{p}_{m2} \log(\hat{p}_{m2})$.

The classification error is $E = 1 - \max_k(\hat{p}_{mk}) = 1 - \max\{\hat{p}_{m1}, \hat{p}_{m2}\}$.

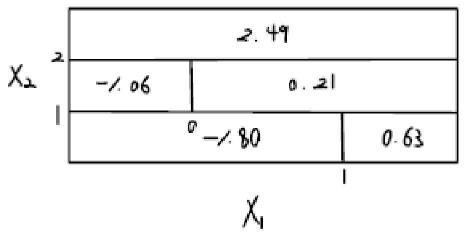


#4.

(a)



(b)

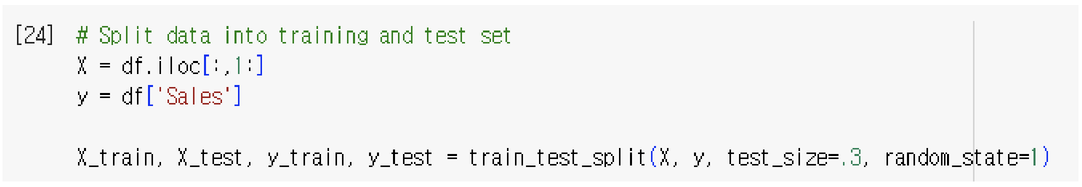


#5.

According to majority vote approach gives 'red' as a result since 6 out of 10 probabilities exceed 0.5, which means that class for given data is likely to be red. However, according to approach based on average probability, it results in 'green', since average probability for then estimates is 0.45, which allocates given data to class 'green'.

#8.

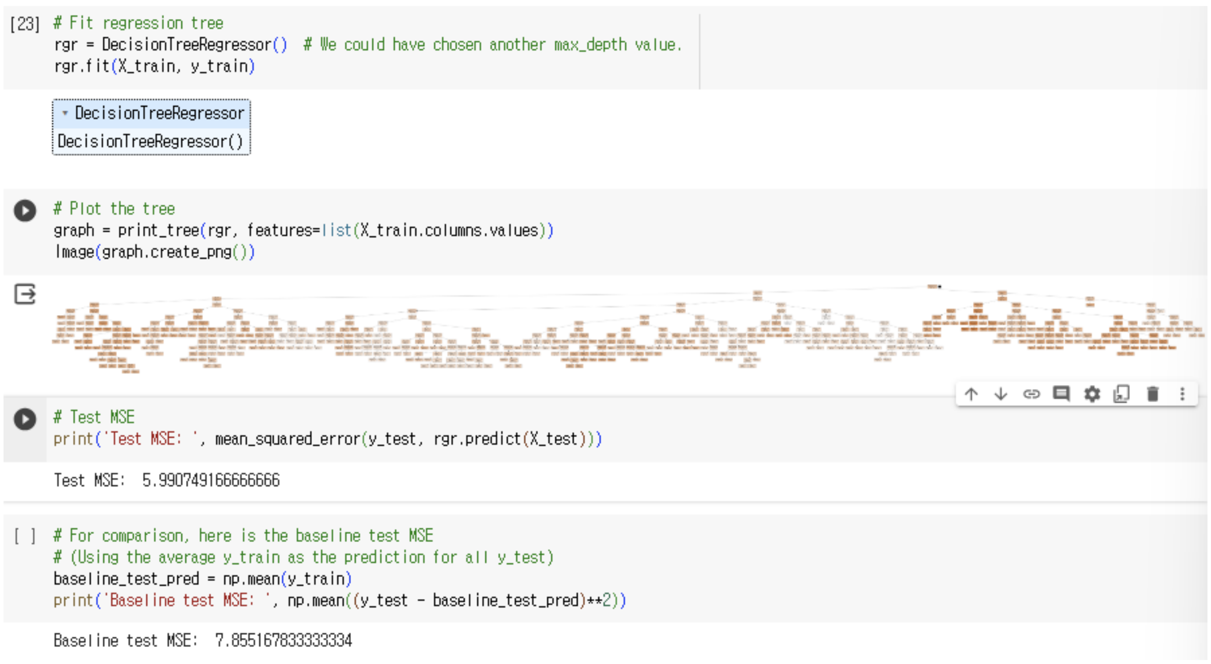
(a)



```
[24] # Split data into training and test set
     X = df.iloc[:,1:]
     y = df['Sales']

     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=1)
```

(1st column is 'Sales', so that X was constructed by excluding 1st column)

(b)

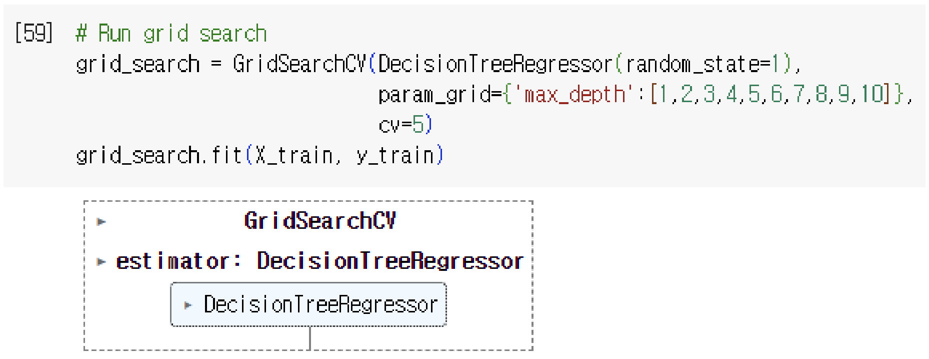


```
[23] # Fit regression tree
     rgr = DecisionTreeRegressor()  # We could have chosen another max_depth value.
     rgr.fit(X_train, y_train)

     ▾ DecisionTreeRegressor
     DecisionTreeRegressor()

     # Plot the tree
     graph = print_tree(rgr, features=list(X_train.columns.values))
     Image(graph.create_png())
```

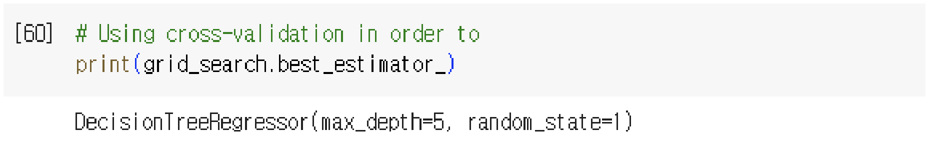
```
     # Test MSE
     print('Test MSE: ', mean_squared_error(y_test, rgr.predict(X_test)))

     Test MSE:  5.990749166666666

[ ]  # For comparison, here is the baseline test MSE
     # (Using the average y_train as the prediction for all y_test)
     baseline_test_pred = np.mean(y_train)
     print('Baseline test MSE: ', np.mean((y_test - baseline_test_pred)**2))

     Baseline test MSE:  7.855167833333334
```

Test MSE obtained from this tree was 5.990749166666666. Although this tree seems too complex, it gives better performance than base test MSE, which was evaluated as residual sum of squares using test mean as predicted values.

(c)



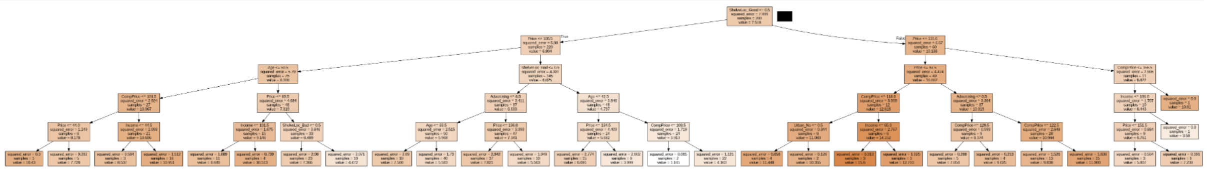
```
[59] # Run grid search
     grid_search = GridSearchCV(DecisionTreeRegressor(random_state=1),
                                param_grid={'max_depth':[1,2,3,4,5,6,7,8,9,10]},
                                cv=5)
     grid_search.fit(X_train, y_train)

     ▸           GridSearchCV
     ▸ estimator: DecisionTreeRegressor
            ▸ DecisionTreeRegressor
```



```
[60] # Using cross-validation in order to
     print(grid_search.best_estimator_)

     DecisionTreeRegressor(max_depth=5, random_state=1)
```

Using cross validation gives maximum depth 5 as the optimal level of convexity when we do not consider pruning. In this case, tree was obtained as follow, and test MSE was 5.564392896935152. It gives better test MSE than before.

```
graph = print_tree(rgr, features=list(X_train.columns.values))
Image(graph.create_png())
```



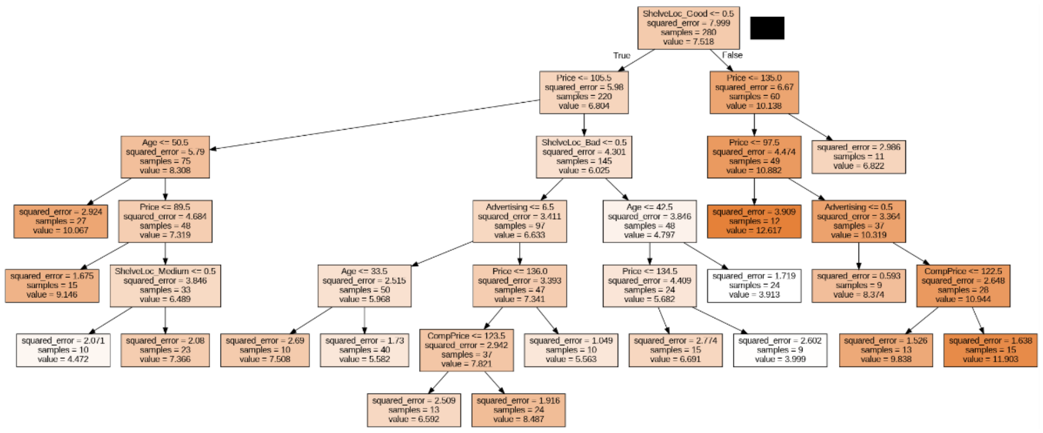
```
# Test MSE before pruning
print('Test MSE: ', mean_squared_error(y_test, rgr.predict(X_test)))
```

```
Test MSE:  5.564392896935152
```

But when we additionally consider pruning, test MSE decreased further into 5.12730232127365. From this result, one can observe pruning improves the performance, as it prevents tree to be overly overfitted, which makes model less explainable in the new observations.

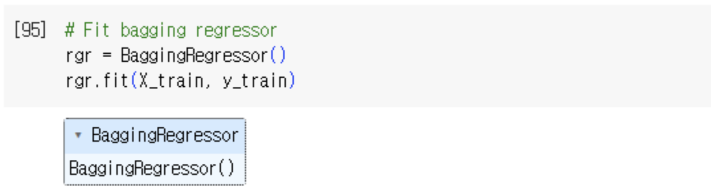
```
[257] pruning_parameters = np.linspace(0, 1, 1001)

      best_mean_squared_error = float('inf')
      best_pruning_param = None

      # Iterate over each pruning parameter and perform cross-validation
      for param in pruning_parameters:
          regressor = DecisionTreeRegressor(ccp_alpha=param)
          # Perform 5-fold cross-validation, you can change the number of folds (cv) as needed
          scores = -cross_val_score(regressor, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
          mean_mse = np.mean(scores)
          if mean_mse < best_mean_squared_error:
              best_mean_squared_error = mean_mse
              best_pruning_param = param

      # Train a decision tree regressor with the best pruning parameter on the entire training set
      best_regressor = DecisionTreeRegressor(ccp_alpha=best_pruning_param)
      best_regressor.fit(X_train, y_train)

      # Evaluate the best model on the test set
      y_pred = best_regressor.predict(X_test)
      test_mse = mean_squared_error(y_test, y_pred)

      print("Best pruning parameter:", best_pruning_param)
      print("Best cross-validation mean squared error:", best_mean_squared_error)
      print("Test set mean squared error:", test_mse)
```

```
Best pruning parameter: 0.105
Best cross-validation mean squared error: 4.824893272941649
Test set mean squared error: 5.12730232127365
```



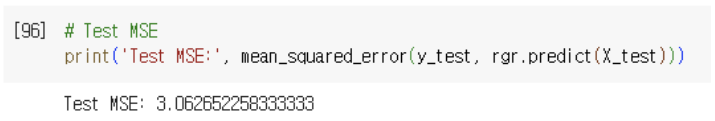
ShelveLoc_Good <= 0.5 squared_error = 7.999 samples = 280 value = 7.518
True
False
Price <= 105.5 squared_error = 5.98 samples = 220 value = 6.804
Price <= 135.0 squared_error = 6.67 samples = 60 value = 10.138
Age <= 50.5 squared_error = 5.79 samples = 75 value = 8.308
ShelveLoc_Bad <= 0.5 squared_error = 4.301 samples = 145 value = 6.025
Price <= 97.5 squared_error = 4.474 samples = 49 value = 10.882
squared_error = 2.986 samples = 11 value = 6.822
squared_error = 2.924 samples = 27 value = 10.067
Price <= 89.5 squared_error = 4.684 samples = 48 value = 7.319
Advertising <= 6.5 squared_error = 3.411 samples = 97 value = 6.633
Age <= 42.5 squared_error = 3.846 samples = 48 value = 4.797
squared_error = 3.909 samples = 12 value = 12.617
Advertising <= 0.5 squared_error = 3.364 samples = 37 value = 10.319
squared_error = 1.675 samples = 15 value = 9.146
ShelveLoc_Medium <= 0.5 squared_error = 3.846 samples = 33 value = 6.489
Age <= 33.5 squared_error = 2.515 samples = 50 value = 5.968
Price <= 136.0 squared_error = 3.393 samples = 47 value = 7.341
Price <= 134.5 squared_error = 4.409 samples = 24 value = 5.682
squared_error = 1.719 samples = 24 value = 3.913
squared_error = 0.593 samples = 9 value = 8.374
CompPrice <= 122.5 squared_error = 2.648 samples = 28 value = 10.944
squared_error = 2.071 samples = 10 value = 4.472
squared_error = 2.08 samples = 23 value = 7.366
squared_error = 2.69 samples = 10 value = 7.508
squared_error = 1.73 samples = 40 value = 5.582
CompPrice <= 123.5 squared_error = 2.942 samples = 37 value = 7.821
squared_error = 1.049 samples = 10 value = 5.563
squared_error = 2.774 samples = 15 value = 6.691
squared_error = 2.602 samples = 9 value = 3.999
squared_error = 1.526 samples = 13 value = 9.838
squared_error = 1.638 samples = 15 value = 11.903
squared_error = 2.509 samples = 13 value = 6.592
squared_error = 1.916 samples = 24 value = 8.487

(d)



```
[95] # Fit bagging regressor
     rgr = BaggingRegressor()
     rgr.fit(X_train, y_train)
```

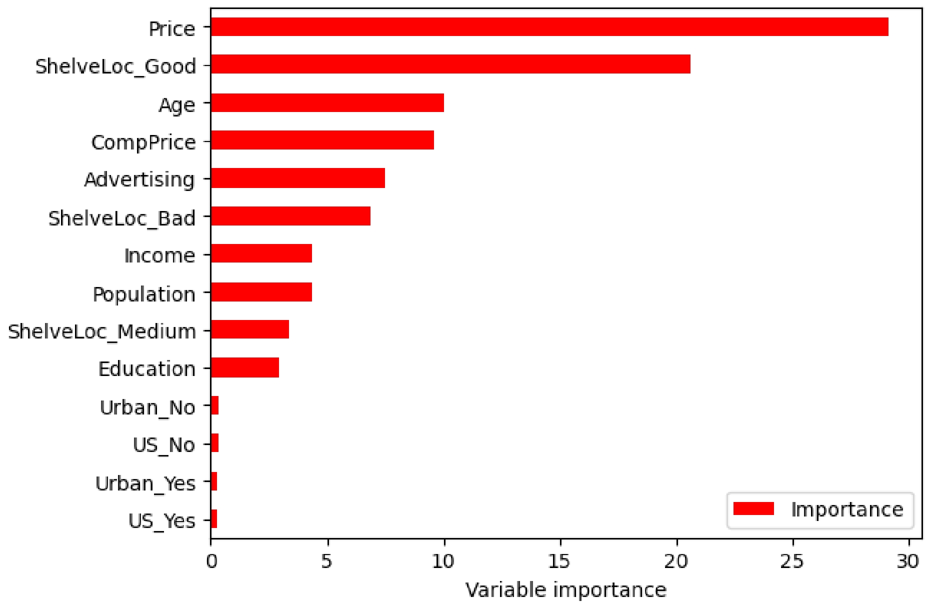
```
▾ BaggingRegressor
BaggingRegressor()
```



```
[96] # Test MSE
     print('Test MSE:', mean_squared_error(y_test, rgr.predict(X_test)))
```

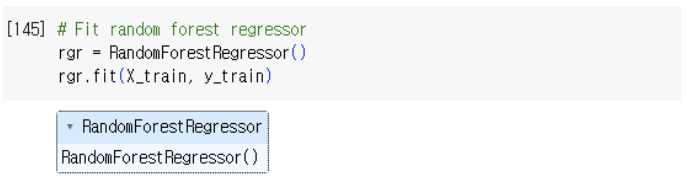
```
Test MSE: 3.062652258333333
```

Using bagging approach gave 3.062652258333333 of test MSE.



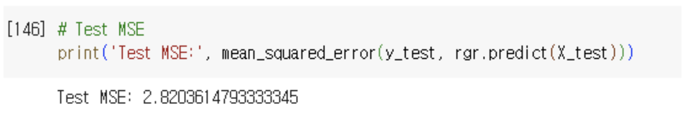
The figure above illustrates the importance of variable predicting response variable. From this result, one can conclude that price is the most important variable in predicting response variable.

(e)



```
[145] # Fit random forest regressor
      rgr = RandomForestRegressor()
      rgr.fit(X_train, y_train)
```

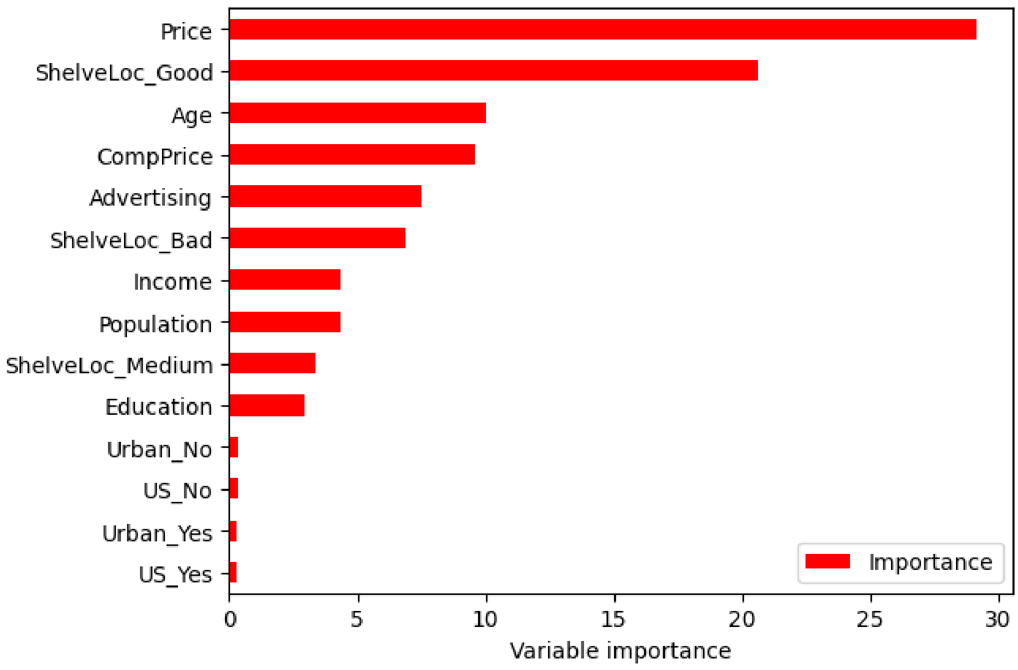
```
▾ RandomForestRegressor
RandomForestRegressor()
```



```
[146] # Test MSE
      print('Test MSE:', mean_squared_error(y_test, rgr.predict(X_test)))
```

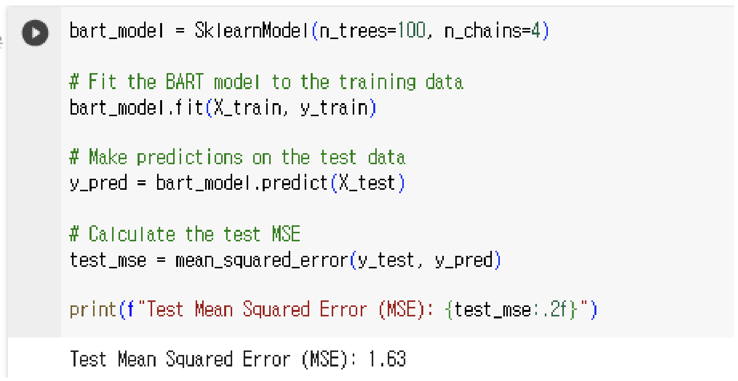
```
Test MSE: 2.8203614793333345
```

Using random forest approach gave 2.8203614793333345 of test MSE.



This figure describes the relative importance between predictors. From this result, one can conclude that price is the most important variable in predicting response variable.

(f)



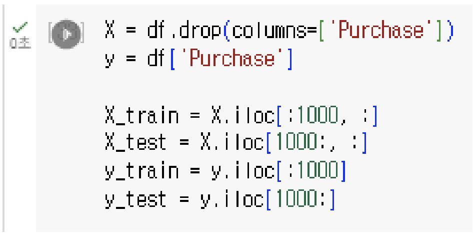
```
bart_model = SklearnModel(n_trees=100, n_chains=4)

# Fit the BART model to the training data
bart_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = bart_model.predict(X_test)

# Calculate the test MSE
test_mse = mean_squared_error(y_test, y_pred)

print(f"Test Mean Squared Error (MSE): {test_mse:.2f}")
```

```
Test Mean Squared Error (MSE): 1.63
```

Using BART model, the lowest test MSE(about 1.63) was obtained.

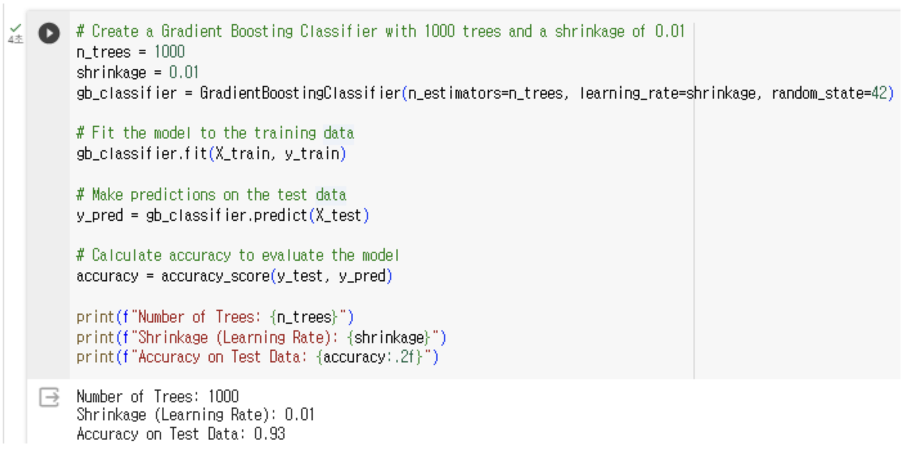
#11.

(a)



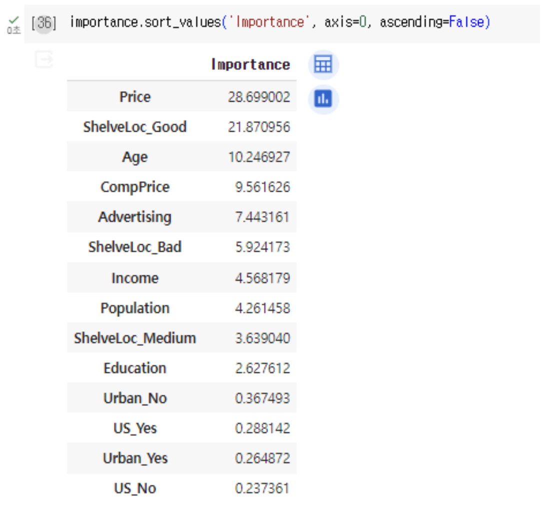
```
X = df.drop(columns=['Purchase'])
y = df['Purchase']

X_train = X.iloc[:1000, :]
X_test = X.iloc[1000:, :]
y_train = y.iloc[:1000]
y_test = y.iloc[1000:]
```

(b)



```
# Create a Gradient Boosting Classifier with 1000 trees and a shrinkage of 0.01
n_trees = 1000
shrinkage = 0.01
gb_classifier = GradientBoostingClassifier(n_estimators=n_trees, learning_rate=shrinkage, random_state=42)

# Fit the model to the training data
gb_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = gb_classifier.predict(X_test)

# Calculate accuracy to evaluate the model
accuracy = accuracy_score(y_test, y_pred)

print(f"Number of Trees: {n_trees}")
print(f"Shrinkage (Learning Rate): {shrinkage}")
print(f"Accuracy on Test Data: {accuracy:.2f}")
```

```
Number of Trees: 1000
Shrinkage (Learning Rate): 0.01
Accuracy on Test Data: 0.93
```

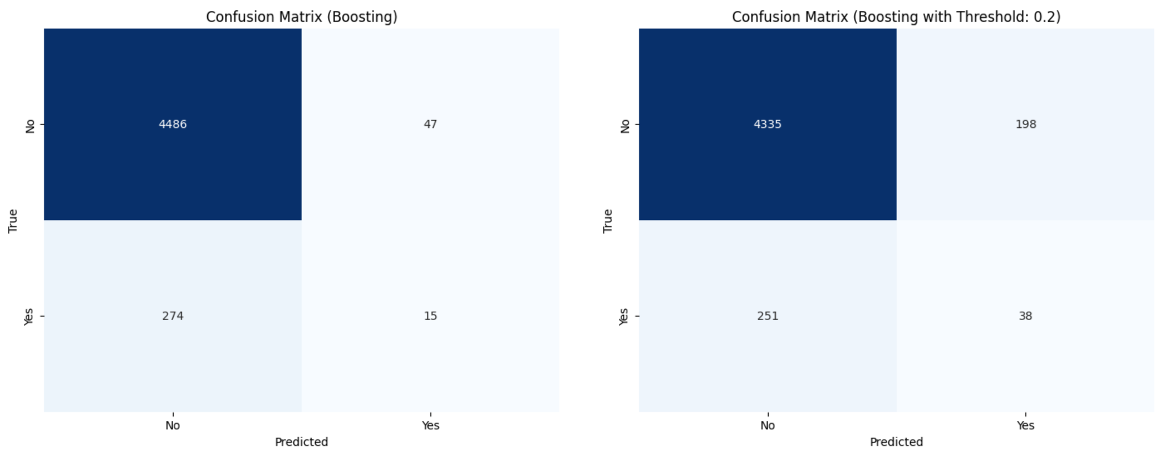


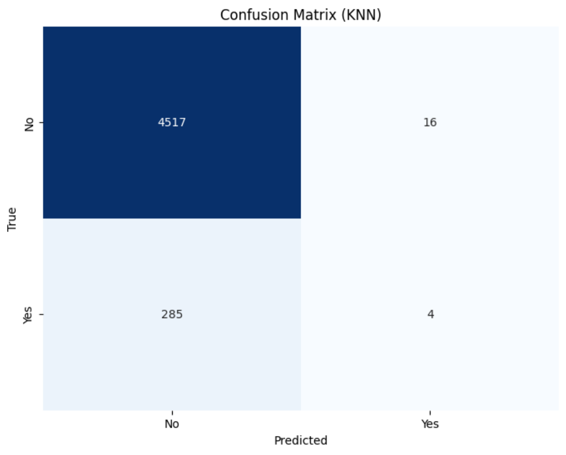
```
[36] importance.sort_values('Importance', axis=0, ascending=False)
```

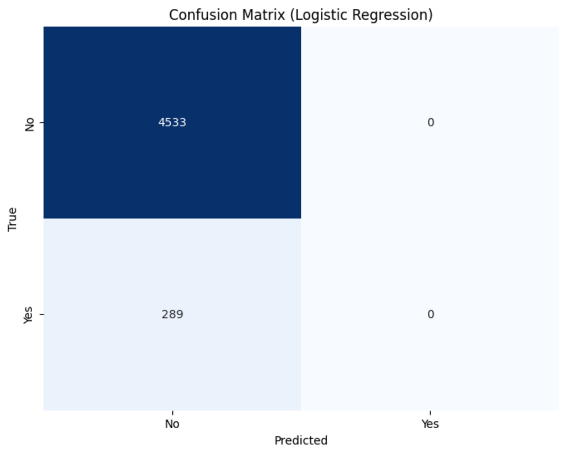
	Importance
Price	28.699002
ShelveLoc_Good	21.870956
Age	10.246927
CompPrice	9.561626
Advertising	7.443161
ShelveLoc_Bad	5.924173
Income	4.568179
Population	4.261458
ShelveLoc_Medium	3.639040
Education	2.627612
Urban_No	0.367493
US_Yes	0.288142
Urban_Yes	0.264872
US_No	0.237361

From a boosting model consists of 1000 trees, and a shrinkage value 0.01, 'Price' resulted into be the most important predictor.

(c)







These four figures illustrate classification result as a form of confusion matrix using Boosting, Boosting with acceptance threshold probability 0.2, KNN, and Logistic Regression respectively.

From this result, the fraction of the people predicted to make a purchase do in fact make from these models are evaluated as below.

- Boosting: $\frac{15}{289}$
- Boosting with acceptance threshold probability 0.2: $\frac{38}{289}$
- KNN: $\frac{4}{289}$
- Logistic Regression: $\frac{0}{289}$

In overall, every method did not perform well in terms of classification. I think this is because there are dominantly many 'No' in the given 'Purchase' data, while there is poor amount of 'Yes' in the data. Therefore, it cannot learn data well.

Chapter11:

#2.

(a) No, Yes, Yes and No. Second participant was the case that study has ended before event occurs, and the third participant was the case that one dropped out of the study.

(b) $c_1 = NA, c_2 = 2, c_3 = 1.5, c_4 = NA$

(c) $t_1 = 1.2, t_2 = NA, t_3 = NA, t_4 = 0.2$

(d) $y_1 = 1.2, t_2 = 2, t_3 = 1.5, t_4 = 0.2$

(e) $\delta_1 = 1, \delta_2 = 1, \delta_3 = 0, \delta_4 = 1$

#3.

$K = 2$: the number of unique replacements

$d_1 = 0.2, d_2 = 1.2$: unique replacement times

$\tau_1 = 4, \tau_2 = 3$: the number of participants who did not replaced phone until event happens

$q_1 = 1, q_2 = 1$: the number of participants who replaced the phone

#4.

(a) There are two 2 events that happen before 50 days. There are 6 data, but as the earliest observation has censored, we start the number of risk to 5. Using this fact, the estimated probability of survival past 50 days can be evaluated as follows.

$$\hat{S}(d_2) = \hat{P}(T > d_2) = \hat{P}(T > d_2|T > d_1)\hat{P}(T > d_1) = \frac{r_2 - q_2}{r_2} \times \frac{r_1 - q_1}{r_1} = \frac{3}{4} \times \frac{4}{5} = 0.6$$

(b) As there are 3 events among uncensored data occurred at a unique time, estimated survival probability can be evaluated as follows, which consists of 4 intervals.

$$\hat{S}(t) = \begin{cases} 1, & t < 26.5 \\ 0.8, & 26.5 \leq t < 37.2 \\ 0.6, & 37.2 \leq t < 57.3 \\ 0.4, & 57.3 \leq t \end{cases}$$

#7.

(a) q_{1k} can be considered as the random variable that chooses q_k people of deaths occurred in group 1 among group 1(r_{1k} people) and group 2($r_k - r_{1k}$ people). Therefore, we can argue q_{1k} follows the hypergeometric distribution as follows.

$$q_{1k} \sim Hypergeo(r_{1k}, r_k - r_{1k}, q_k)$$

(b) It is well known fact that a random variable which follows hypergeometric distribution has mean and variance as below.

$$X \sim Hypergeo(N, M, n)$$

$$E[X] = \frac{nN}{N+M}, Var(X) = \frac{nNM}{(N+M)^2}\left(1 - \frac{n-1}{N+M-1}\right)$$

Using this fact, we can evaluate the mean and variance of q_{1k} as follows.

$$E[q_{1k}] = \frac{q_k r_{1k}}{r_{1k} + r_k - r_{1k}} = \frac{r_{1k}}{r_k} q_k,$$

$$Var(q_{1k}) = \frac{q_k r_{1k}(r_k - r_{1k})}{(r_{1k} + r_k - r_{1k})^2}\left(1 - \frac{q_k - 1}{r_{1k} + r_k - r_{1k} - 1}\right) = \frac{q_k \left(\frac{r_{1k}}{r_k}\right)\left(1 - \frac{r_{1k}}{r_k}\right)(r_k - q_k)}{r_k - 1}$$

It corresponds to the result of (11.5) and (11.6) in the textbook

#8.

As $S(t) = \Pr(T > t)$, $F(t) = 1 - S(t) = \Pr(T \leq t)$ is a cdf of T.

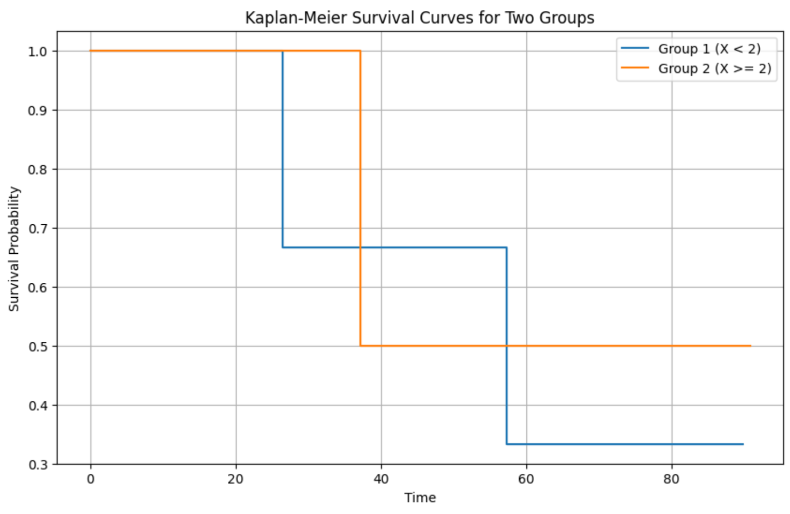
$\therefore f(t) = \frac{dF(t)}{dt} = \Pr(T = t)$.

Then, $h(t) = \frac{f(t)}{S(t)} = \frac{\frac{dF(t)}{dt}}{S(t)} = -\frac{\frac{dS(t)}{dt}}{S(t)}$. Integration from 0 to t gives $\int_0^t h(u)\,du = \int_0^t -\frac{\frac{dS(u)}{du}}{S(u)}du = -\ln S(t)$.

$$\therefore S(t) = \exp\left(-\int_0^t h(u)\,du\right)$$

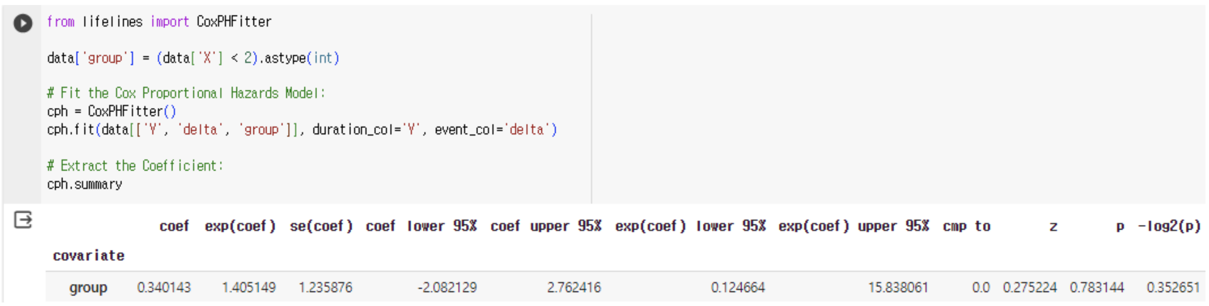
#11.

(a)



It does not seem to have similarity in terms of survival curves between two groups.

(b)

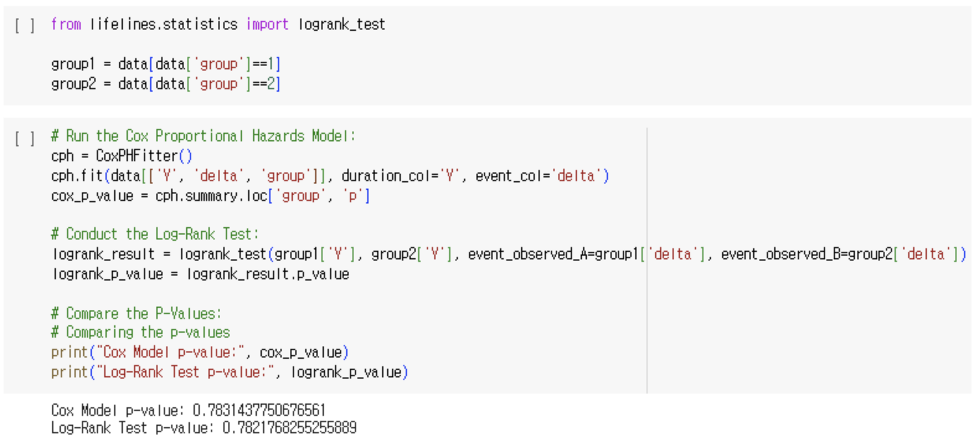


```
from lifelines import CoxPHFitter

data['group'] = (data['X'] < 2).astype(int)

# Fit the Cox Proportional Hazards Model:
cph = CoxPHFitter()
cph.fit(data[['Y', 'delta', 'group']], duration_col='Y', event_col='delta')

# Extract the Coefficient:
cph.summary
```

```
               coef  exp(coef)  se(coef)  coef lower 95%  coef upper 95%  exp(coef) lower 95%  exp(coef) upper 95%  cmp to         z         p  -log2(p)
covariate
    group  0.340143   1.405149  1.235876       -2.082129        2.762416             0.124664            15.838061     0.0  0.275224  0.783144  0.352651
```

The coefficient is 0.34. This means that there exists an increased hazard with group having covariate 'X' less than 2. However, there is no evidence that the true coefficient value is non-zero, because the p-value of it is 0.78.

(c)



```python
from lifelines.statistics import logrank_test

group1 = data[data['group']==1]
group2 = data[data['group']==2]
```

```python
# Run the Cox Proportional Hazards Model:
cph = CoxPHFitter()
cph.fit(data[['Y', 'delta', 'group']], duration_col='Y', event_col='delta')
cox_p_value = cph.summary.loc['group', 'p']

# Conduct the Log-Rank Test:
logrank_result = logrank_test(group1['Y'], group2['Y'], event_observed_A=group1['delta'], event_observed_B=group2['delta'])
logrank_p_value = logrank_result.p_value

# Compare the P-Values:
# Comparing the p-values
print("Cox Model p-value:", cox_p_value)
print("Log-Rank Test p-value:", logrank_p_value)
```

```
Cox Model p-value: 0.7831437750676561
Log-Rank Test p-value: 0.7821768255255889
```

They are identical.