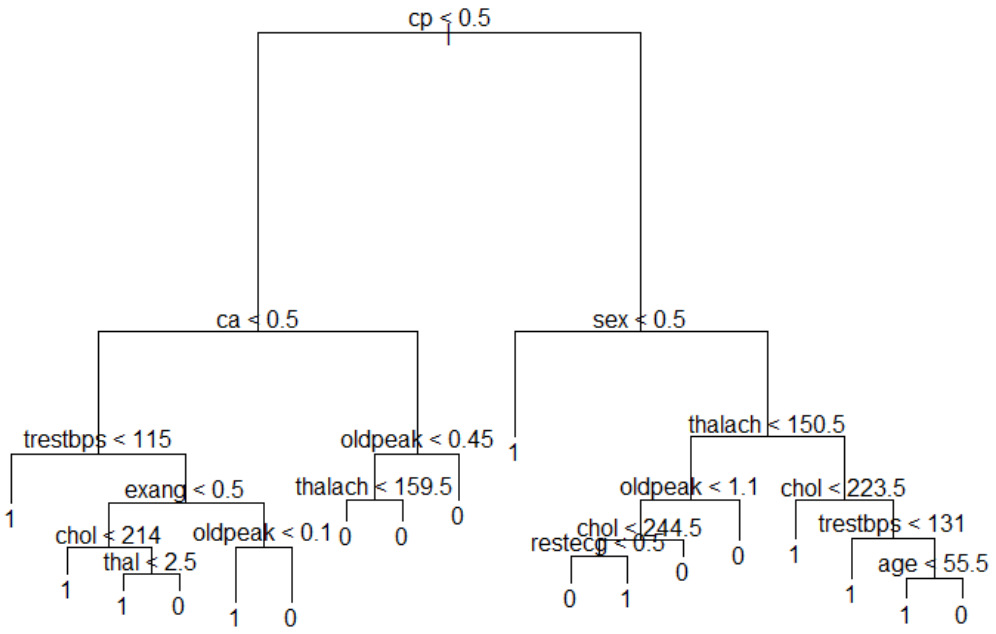
<다변량분석 과제 #6>

2014170852 산업경영공학부 조영관

전체 데이터셋을 200개의 training set과 103개의 validation set으로 분할하였다.

<Tree package를 이용해 Classification Tree 학습하기 (Pruning 하지 않은 상태)>

Classification Tree를 학습한 결과는 다음과 같다.



변수의 특정 값 기준에 따라 세부적으로 분류되었음을 확인할 수 있다.

예를 들어, cp<0.5가 yes이면 왼쪽으로 가지가 뻗고, 그렇지 않다면 오른쪽으로 가지를 뻗는다.

Cp<0.5인 관측치들 중 ca<0.5 이면 또 왼쪽으로 가지가 뻗는다. 그리고 그 관측치 중 trestbps<115가 yes이면 1로 분류된다. 즉, heart disease가 존재하는 환자로 구분할 수 있다.

다른 것들도 이런 방식으로 해석하면 된다.

```
Classification tree:
tree(formula = train$data.target ~ ., data = train)
Variables actually used in tree construction:
 [1] "cp"       "ca"       "trestbps" "exang"    "chol"     "thal"     "oldpeak"
 [8] "thalach"  "sex"      "restecg"  "age"
Number of terminal nodes:  18
Residual mean deviance:  0.3592 = 65.37 / 182
Misclassification error rate: 0.08 = 16 / 200
```

Summary를 살펴보면, 위 tree를 만드는데 사용된 주요 변수들이 나열된다.

2개의 변수 빼고 11개의 입력변수들이 사용되었음을 확인할 수 있다.

Cp, ca, testbps, exang, chol, thal, oldpeak thalach, sex, restecg, age 이렇게 11개다.

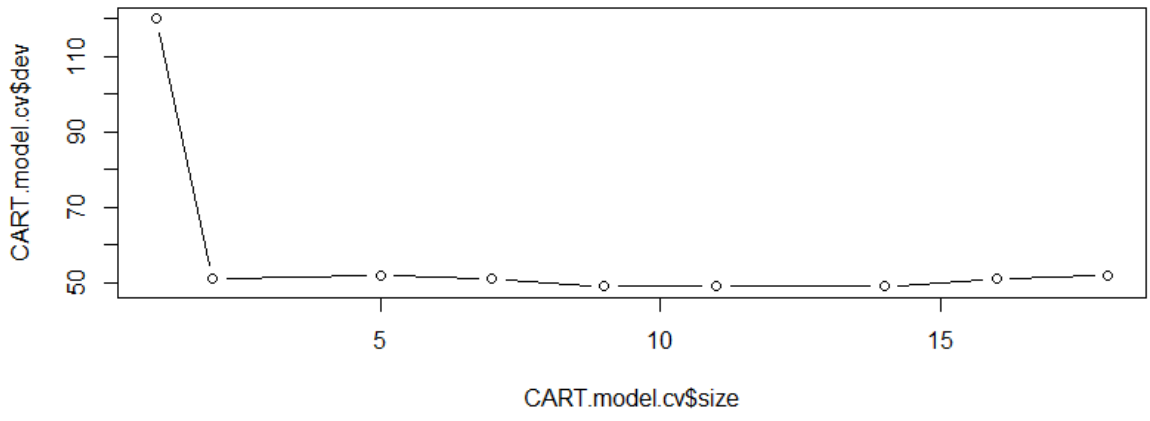
분류 성능을 확인해보자.

```
  CART.prey.full
    0  1
0 28 11
1 11 53
```

```
                TPR Precision       TNR  Accuracy       BCR F1-Measure
tree.full  0.828125  0.828125 0.7179487 0.7864078 0.7710715  0.828125
```

위 결과를 보면, Recall(재현율)과 Precision(정밀도)값이 0.828 정도임을 확인할 수 있으며, F1 값이 0.828, BCR값이 0.77임을 확인할 수 있다. 성능이 좋음을 확인할 수 있다.

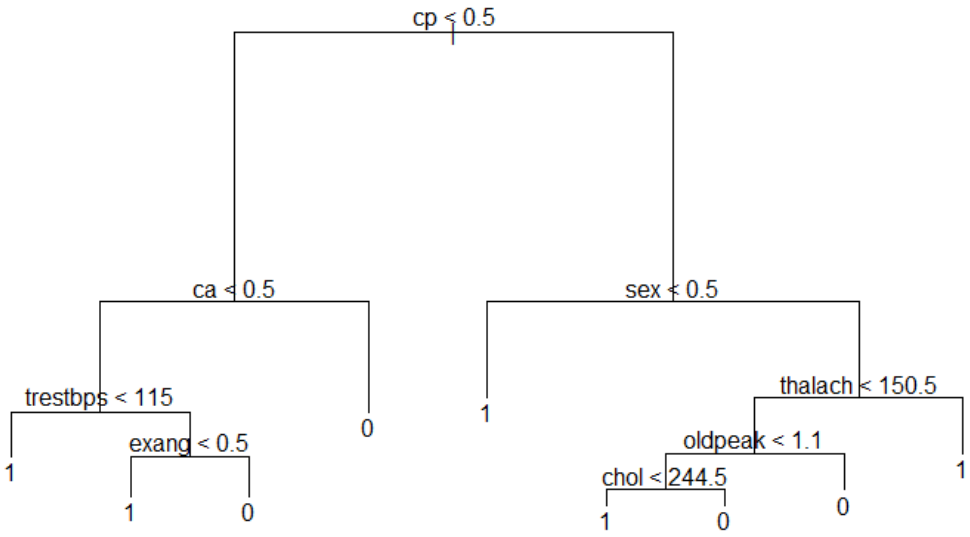
<Pruning을 진행한 후의 Tree 결과>



Tree size (terminal node 수) 변화에 따른 deviance의 변화를 확인해 보았다.

그 결과 값이 9일 때 최소 값을 가짐을 확인할 수 있었다.

따라서 tree size (terminal node 수)를 9로 조정하여 (Pruning) 결과물을 plotting 해보았다.



가지치기를 하지 않았을 때보다 더 간단해진 tree의 모습을 확인할 수 있다.

출력물로 나오는 size도 9임을 확인할 수 있다.

해석은 위의 가지치기 전의 tree와 동일하다. Cp<0.5가 yes이면 왼쪽으로 이동하고, 그렇지 않으면 오른쪽으로 이동한다. Cp<0.5인 관측치 중 ca<0.5이면 왼쪽으로 이동한다. Ca<0.5중 trestbps<115가 yes면 1로 분류된다. 즉, 심장병을 가지고 있는 환자로 분류된다.

다른 결과들도 비슷하게 해석하면 된다.

```
Classification tree:
snip.tree(tree = CART.model, nodes = c(5L, 19L, 56L, 15L, 18L
))
Variables actually used in tree construction:
[1] "cp"       "ca"       "trestbps" "exang"    "sex"      "thalach"  "oldpeak"
[8] "chol"
Number of terminal nodes:  9
Residual mean deviance:  0.6538 = 124.9 / 191
Misclassification error rate: 0.125 = 25 / 200
```

Summary를 살펴보자.

위 tree 학습에 실질적으로 사용된 변수는 8개이다.

Cp, ca, trestbps, exang, sex, thalach, oldpeak, chol 이렇게 8가지 변수가 실질적으로 tree 구축에 사용되었다. 더 적은 변수가 사용되었음을 확인할 수 있다.

오분류율은 약 0.125가 나왔음을 확인할 수 있다.

성능을 확인해보자.

```
  CART.prey
    0  1
0 27 12
1 12 52
```

```
                TPR Precision       TNR  Accuracy       BCR F1-Measure
tree.full  0.828125  0.828125 0.7179487 0.7864078 0.7710715   0.828125
tree.prune 0.812500  0.812500 0.6923077 0.7669903 0.7500000   0.812500
```

위 confusion matrix를 확인해보면, 실제 심장병이 있는데 심장병으로 예측하지 않은 개수가 12개가 된다.

아래 지표 결과값을 보자. (tree.prune을 확인하면 된다.)

즉, 재현율이 0.8125다.

정밀도 값도 0.8125로 동일하다. 그리고 BCR값은 0.75이며 F1값이 0.8125임을 알 수 있다.

가지치기를 하지 않았을 때보다, 성능 지표 값은 떨어지지만, 크게 감소하지 않았고 분류도 시각적으로 명확하므로 더 좋다고 말할 수 있다.

<Rpart package를 이용한 Classification tree 학습 진행>

사용한 패키지의 이름은 'rpart' 이다.

```
rpart(formula, data, weights, subset, na.action = na.rpart, method,
      model = FALSE, x = FALSE, y = TRUE, parms, control, cost, ...)
```

Rpart 함수의 옵션은 위와 같다.

이 중 weights는 필요한 경우 변수에 무게를 부여하는 옵션이며,

Method의 경우 anova, poission, class, exp 중 하나를 지정할 수 있다.

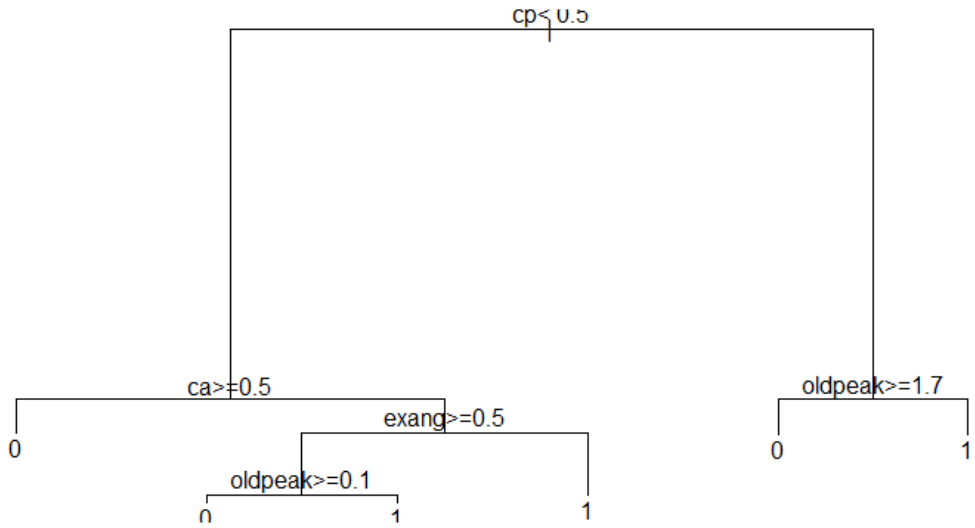
Y가 survival object이면 exp를 쓰며, 2 column이면 poisson을 쓰고 y가 factor이면 class를 쓰며 다른 경우는 anova를 사용한다.

Parms는 splitting function을 지정해주는 옵션이다. Anova는 parameter가 필요 없으며, poission은

단일 파라미터가 필요하고, exp는 poisson과 동일하고, 분류 split의 경우 사전확률의 벡터들 모두 포함 가능하다.

Splitting index는 gini 또는 information 둘 중 하나로 지정할 수 있다.

Rpart package를 이용해 도출한 tree 결과를 보자.



Tree package 결과와는 다르게 terminal node 값이 6개로 추가적인 pruning을 진행하지 않아도 잘 분류된 것으로 보인다.

```
  rpart.prey
    0  1
 0 29 10
 1 12 52
```

	TPR	Precision	TNR	Accuracy	BCR	F1-Measure
tree.full	0.828125	0.8281250	0.7179487	0.7864078	0.7710715	0.8281250
tree.prune	0.812500	0.8125000	0.6923077	0.7669903	0.7500000	0.8125000
rpart.full	0.812500	0.8387097	0.7435897	0.7864078	0.7772816	0.8253968

Rpart 패키지를 이용해 tree를 학습한 결과는 다음과 같다.

Recall은 0.8125, Precision은 0.8387이다. BCR은 0.777이며 F1은 0.8254이다.

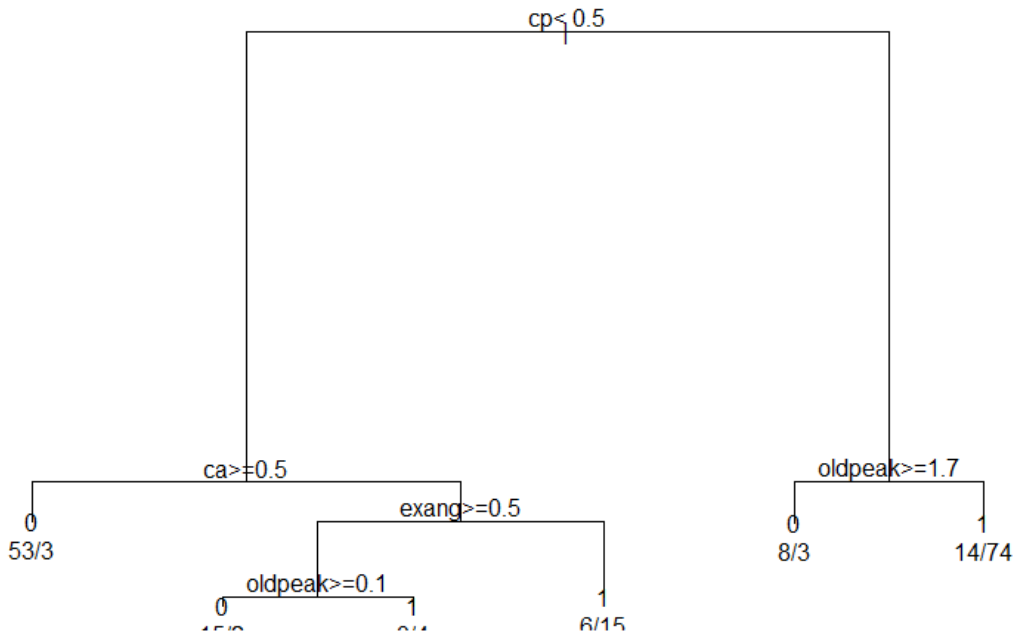
Tree package의 Prune된 tree보다 학습이 좋음을 확인할 수 있다.

Rpart에는 splitting index 두 가지를 이용할 수 있다.

Gini와 information이 그 두가지다.

각각의 index를 활용해 tree를 학습해보자.

먼저 gini index를 사용한 결과이다.



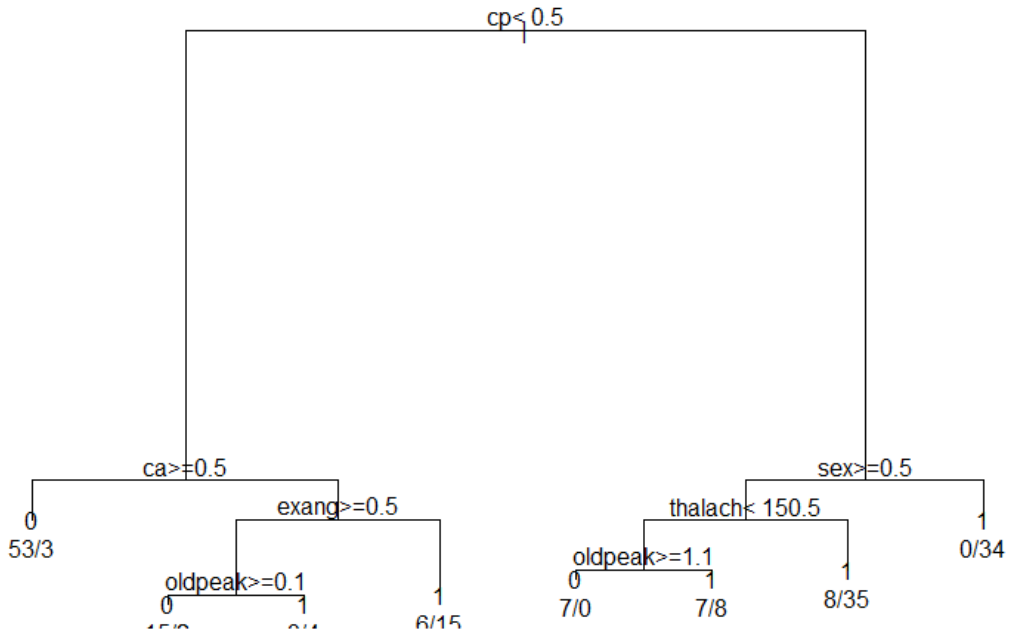

```
  rpartpredgini
    0  1
0 29 10
1 12 52
```

Confusion matrix는 위와 같으며 성능 지표는 아래와 같다.

```
                TPR Precision       TNR  Accuracy       BCR F1-Measure
rpart.gini 0.812500 0.8387097 0.7435897 0.7864078 0.7772816  0.8253968
```

Rpart 기본 default index가 gini 인 것으로 보인다.

다음은 information index를 사용한 결과이다.



위 tree 결과를 보면 terminal node 수가 8개임을 확인할 수 있으며, 각각의 0과 1이 어떻게 분류되었는지를 확인할 수 있다.

예를 들어 53/3은 53개가 0으로 분류되고, 3이 1로 분류된 개수이다.

이에 따라 과반수가 넘는 쪽으로 분류가 정해진다.

성능을 확인해보자.

```
  rpartpredinfo
    0  1
0 28 11
1 10 54
```

Confusion matrix는 위와 같으며 성능 지표는 아래와 같다

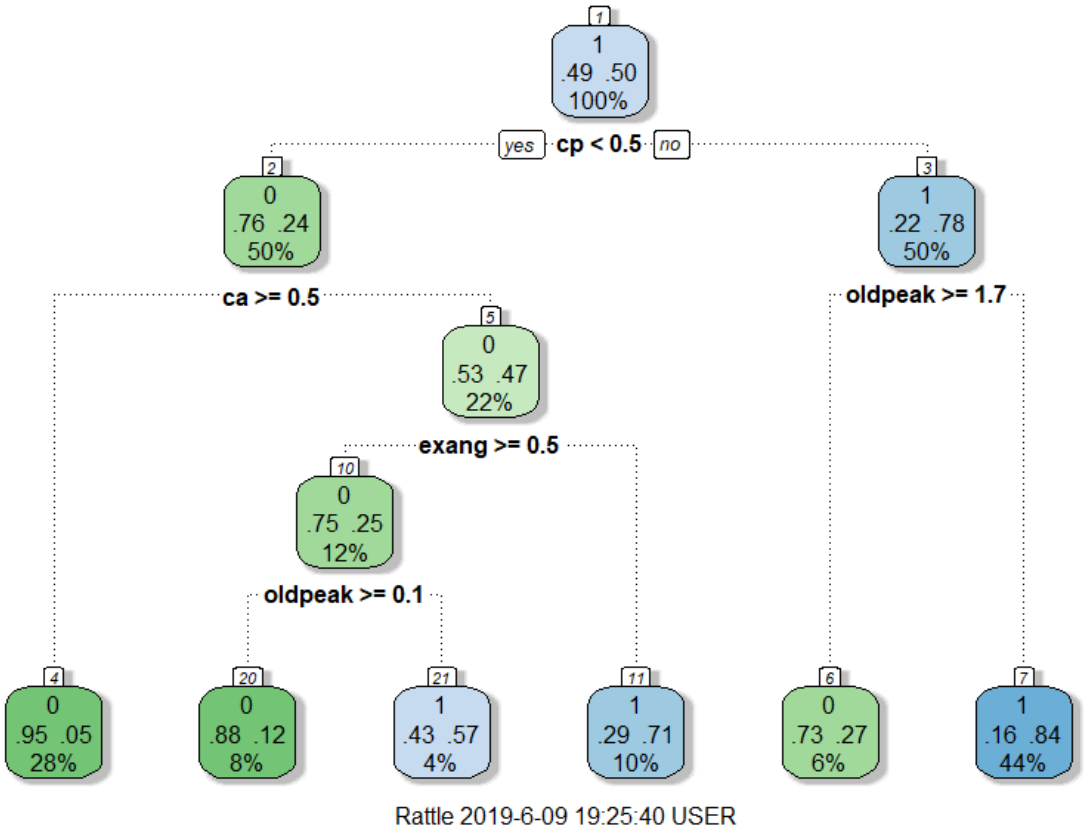
```
                TPR Precision       TNR  Accuracy       BCR F1-Measure

rpart.info 0.843750 0.8307692 0.7179487 0.7961165 0.7783118  0.8372093
```

분류가 더 자세하게 되어서, 성능이 gini와 비교해 높게 나왔다.

따라서 위 데이터셋에 대해서는 gini index보다 information index를 적용하는 것이 더 적합함을 알 수 있다.

따라서 바로 위의 트리가 rpart package에서 best model이다.

Rpart의 tree를 조금 더 명확하게 시각화한 plot을 보자.



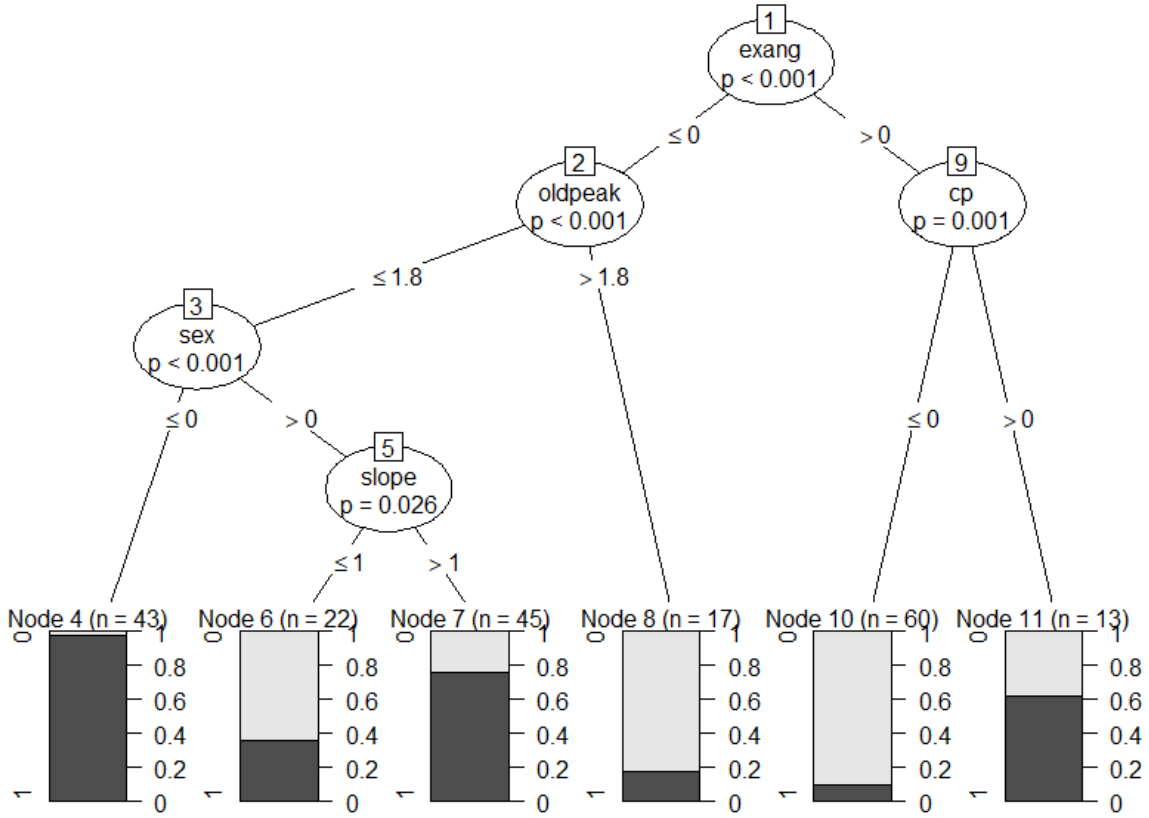
기존의 tree나 rpart 패키지의 기본 plot과 다르게 더 명확하고, 보기 좋게 시각화 되었음을 확인할 수 있다. 기준과 분류 결과만 보여주는 tree plot과 rpart plot과 달리 위의 plot은

각각의 0과 1의 비율과 전체 데이터셋에서 어느 정도의 부분을 차지하는지도 알려준다.

예를 들어, 7번 node는 16%가 0이며, 84%가 1로 분류되고 이 7번 노드에는 전체에서 44%의 데이터셋이 포함되어 있음을 알려준다.

<Party package를 이용한 Classification tree 학습>

Party package를 이용해 classification tree를 학습했다. 그 결과 다음과 같았다.



Tree package나 rpart package와는 달리 첫 번째 root node가 exang으로 시작한다.

Exang이 0보다 크고 cp가 0보다 작으면 대부분이 0으로 분류된다.

반면 exang이 0 이하이고, oldpeak이 1.8 이하이고 sex가 0 이하면 대부분의 관측치들이 1로 분류가 된다. 위 plot을 보면 알 수 있듯이, 다른 tree보다 비교적 분류 성능이 떨어지는 것을 알 수 있다.

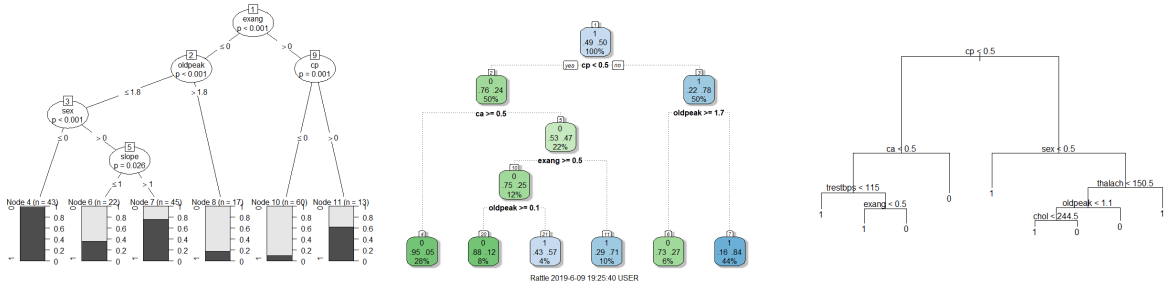
```
   party.prey
     0  1
  0 26 13
  1 20 44
```

```
                TPR Precision       TNR  Accuracy       BCR F1-Measure
party      0.687500 0.7719298 0.6666667 0.6796117 0.6770032   0.7272727
```

분류 성능 지표를 살펴보면, 다른 tree package를 사용한 것에 비해 분류 성능이 떨어짐을 확인할 수 있다. Party package는 이미 pruning 과정을 거치므로 추가적인 pruning을 진행하지 않는다.

<총 정리 및 package 간 plot 비교>

결론적으로 각 package의 plot을 합쳐서 살펴보자.



왼쪽부터 순서대로 party, rpart, tree 패키지이다.

Party는 각 0과 1의 분포를 마지막 terminal node에서 시각화 하여 잘 보여준다.

Rpart는 모든 node에서 0과 1의 분포를 살필 수 있으며, 그 node에 전체 중 몇 개의 데이터셋이 분포 되어있는지 비율을 통해 확인할 수 있다.

Tree는 매우 단순하게, 분류 기준만 보여주며, terminal node에서 어떤 결과로 분류되었는지를 보여준다.

각각의 성능도 살펴보자.

	TPR	Precision	TNR	Accuracy	BCR	F1-Measure
tree.full	0.828125	0.8281250	0.7179487	0.7864078	0.7710715	0.8281250
tree.prune	0.812500	0.8125000	0.6923077	0.7669903	0.7500000	0.8125000
rpart.full	0.812500	0.8387097	0.7435897	0.7864078	0.7772816	0.8253968
rpart.gini	0.812500	0.8387097	0.7435897	0.7864078	0.7772816	0.8253968
rpart.info	0.843750	0.8307692	0.7179487	0.7961165	0.7783118	0.8372093
party	0.687500	0.7719298	0.6666667	0.6796117	0.6770032	0.7272727

rpart의 information index를 활용해 돌렸을 때 제일 높게 나왔음을 확인할 수 있다.

그리고 prune 되기 전이 성능이 더 좋음을 확인할 수 있는데, 이는 더 세부적으로 분류한 것이므로 당연한 결과이지만, 과적합의 문제가 있으니 가지치기는 필수적이다. 따라서 prune된 모델을 봤을 때 성능이 매우 조금 하락하였으므로 prune된 모델이 더 적합함을 알 수 있다.

그리고 gini index보다는 information index를 사용하는 것이 이 데이터셋에 더 적합함을 알 수 있다. 마지막으로 party package는 위의 데이터셋에 적합하지 않은 성능을 보임을 확인할 수 있다.

따라서 이 데이터셋은 rpart package를 이용하고 information index를 사용한 decision tree를 사용하는 것이 제일 좋은 성능을 보임을 알 수 있다.