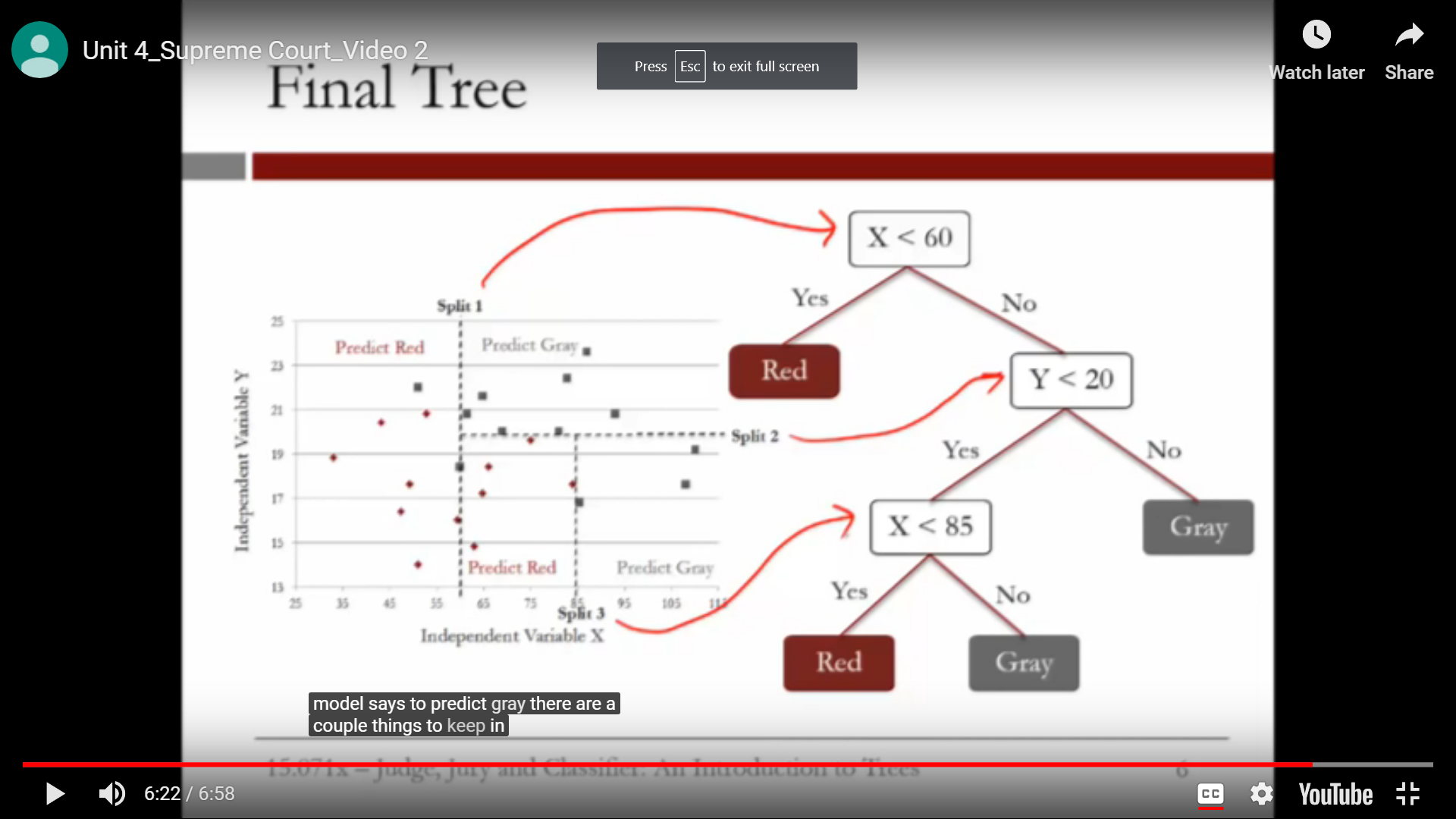
* Using logistic regression, it is sometimes difficult to decide which independent variables are significant and also to predict the outcome for some cases.
* So we use classification and regression tree model where we construct trees and predict the binomial outputs(ie., either 0 or 1)
  + Build a tree by splitting on variables
  + To predict the outcome for an observation, follow the splits and at the end, predict the most frequent outcome.
  + Main advantage is it Does not assume linear model like linear and logistic model and is highly interpretable.
  + In trees, the yes outcomes should also always be on left and the no on the right.



* + We can control how many splits to be generated in various ways.
    - One way is to set lower bound for no of observations in each splitted subset.
    - In R, we have “minbucket” parameter to do this.
      * The smaller it is, the more splits will be generated.
      * If too small, overfitting will occur
      * If too large, model will be too simple with less accuracy.
    - Every subset whould have oberservations of both outcomes.
    - So, we compute the % of data for each output type in that subset.
    - Similar to logistic model, we use threshold value to pick most frequent outcome.
  + To build a tree:
    - install.packages(“rpart”)
    - library(rpart)
    - install.packages(“rpart.plot”)
    - library(rpart.plot)
    - Tree = rpart(dependent\_Variable ~ independent\_variables , data = datset\_name, method= “class”, minbucket = 25)
      * method ===> to tell to build a classificaton tree instead of regression
    - prp(tree\_name) ⇒ to print the treee
  + To predict the model
    - Predict\_cart = predict(tree\_name, newdata = Test, type =”class”)
  + Confusion matrix:
    - table(Test$dependent\_Variable, Predict\_cart)
  + To plot ROC Curve:
    - library(ROCR)
    - PredictROC = predict(Tree\_name, newdata = Test)
    - Pred = prediction(PredictROC[,2], Test$Reverse)
      * predictROC[,2] ⇒ tells to use second column
    - Pref = performance(pred, “tpr”, “fpr”)
      * Tpr ⇒ true possibility rate
      * Fpr ⇒ false possibility rate
    - plot(pref)
* **RANDOM FORESTS:**
  + Similar to CART, designed to improve accuracy of CART prediction.
  + Works **by building large no of cart trees** which makes its less **interpretable**
  + To make a **prediction for new observation**, **each tree “votes”** on the outcome, and **we pick the outcome** that receives **majority of votes**.
  + Each tree can split on only a random subset of the variables with replacement(ie., the obseravtions may repeat in a subset).
  + The subset size can be controlled by parameter **“nodesize”**
  + And no of trees can be controlled by **ntree** parameter
  + TO build the randomForest tree
    - install.packages(“RandomForest”)
    - library(RandomForest)
    - Random forest can also be implemented on regression. But there is no method argument for it to indicate whether it is a classification or regression.
    - As we are going to build a classification tree, we need to convert the outcome variable into factors.
      * dataset\_name$variable\_name = as.factor(dataset\_name$variable\_name)
    - Forest = randomForest(dependent\_variable ~independent\_variable, data = train, nodesize = 25, ntree = 100)
    - PredictForest = predict(forest, newdata = Test)
    - To calculate the accuracy of the test results:
      * table(Test$outcome\_variable, Predcitforest)
* **OCT(Optimal Classification Trees):**
  + The model which can have both high interpretablity and performance.
  + CART tree splits are only locally optimal.
  + To improve the optimizations, they introduced OCT that uses modern optimizations to train the entire data at once instread of split - by - split.
  + Variants of OCT:
    - **OCT:** trees with parallel splits(one variable per split)
    - **OCT-H:** tree with hyperplane splits( multile variables per split)
  + ALso computationally speed
* To select the best value for minbucket prameter:
  + We use **K-fold cross validation**
  + Splits the dataset into k parts( k = 5)
  + We use k-1 folds to estimate the model
  + And remaining one fold to test model.
  + Repeat the above two steps for each of the k folds.
  + From these we select the best parameter that gives high accuracy.
* When we use crossvalidation, we use **complexity-Parameter(CP)** instead of minbucket
  + Smaller cp ⇒ bigger tree(might overfit)
* To use crossvalidation:
  + install.packages(“caret”)
  + library(caret)
  + install.packages(“e1071”)\
  + library(e1017)
  + numOfFolds = trainControl(method = “cv”, number =10)
    - Cv = crossvalidation
  + To get the possible value for cp
    - cpGrid = expand.grid(.cp=seq(0.01, 0.5, 0.01))
  + train(dependent\_variable ~ independent\_variables, data = Train, method = “rpart”, trControl = numFolds, tuneGrid = cpGrid)
  + TreeCV = rpart(dependent\_Variable ~ independent\_variables , data = Train, method= “class”, cp = 0.18)
  + PredictCV = predict(TreeCV, newdata =Test, type =”class”)
  + table(Test$Reverse, PredictCV) ⇒ confusion matrix
* To calculate Penality air:
  + penalityMatrix = matrix(c(0,1,2,3,4,2,1,2,3,4,5,6,1,3,2,4,5,3,1,2,3,4,5,2,3,1,4,2), byrow = TRUE, nrow = 5)
  + Confusionmatrix = as.matrix(table(claimsTest$output, predictmodel))
  + Penality air = sum(confusionmatrix \* penalitymatrix)/nrow(dataset)
* CP(Complexity Parameter): used for tree with no splits
  + Cp = lambda/RSS(no splits)
  + To build a tree with min no of splits:
    - sum(RSS at each leaf) + lambda \* S
      * Lambda ⇒ penality
      * S ⇒ no of splits
  + CP is used to minimise the decision trees and select the optimised tree.