|  |
| --- |
| UMUC |
| Cars Classification via Decision Tree |
| Week 4 Exercise |
|  |
| **DBST 667** |
|  |

|  |
| --- |
| In this exercise, you will use a conditional inference tree (cteee) method to build a classification model for predicting the car’s risk level indicator called symboling based on the car model, fuel type, mileage per hour, number of doors, engine location, etc. You will use the training set to build the model and the test set to evaluate the accuracy of a model. |

Contents

[Cars Dataset 2](#_Toc450414531)

[Launch the Program 2](#_Toc450414532)

[Running the ctree Method 4](#_Toc450414533)

[Divide Data into Training Set and Test Set 4](#_Toc450414534)

[Use the Training Set to Build the Model 5](#_Toc450414535)

[Check the Classification Accuracy for the Training data 8](#_Toc450414536)

[Evaluate the Model for the Test Data 9](#_Toc450414537)

[Troubleshooting 10](#_Toc450414538)

Cars Classification via Conditional Tree

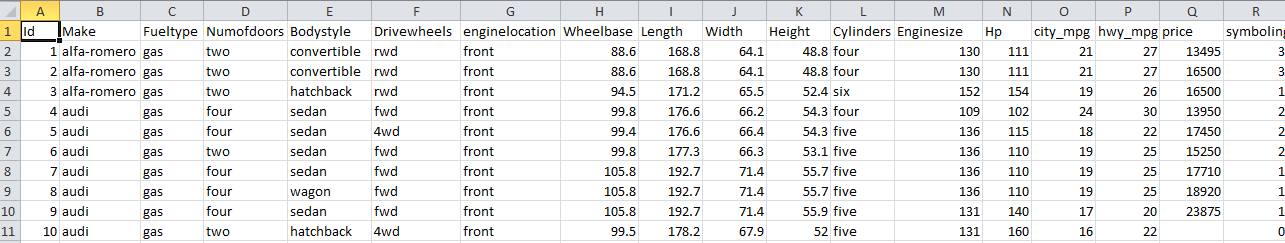
The exercise bellow illustrates the conditional inference tree (ctree) algorithm, which is R implementation of decision tree model. The method can be used for both classification and prediction studies.

In this exercise, we use ctree method to build a classification model for predicting the car’s risk level indicator called symboling based on the car model, fuel type, mileage per hour, number of doors, engine location, etc. The cars with the symboling class +3 are the most risky, while the cars with the symboling lass -3 are the safest. Then, we evaluate the accuracy of the model on test data.

# Cars Dataset

Figure 1 shows the partial content of the cars.csv file. The column headings in the first row of the file are the cars attribute names called variables. The remaining 205 rows are the data, where each row is a single car record.

Column headers are the variable names



Each data row is a car record

Figure 1- Partial Content of cars.csv file

# Launch the Program

Launch the R studio program to open an interface on Figure 2.

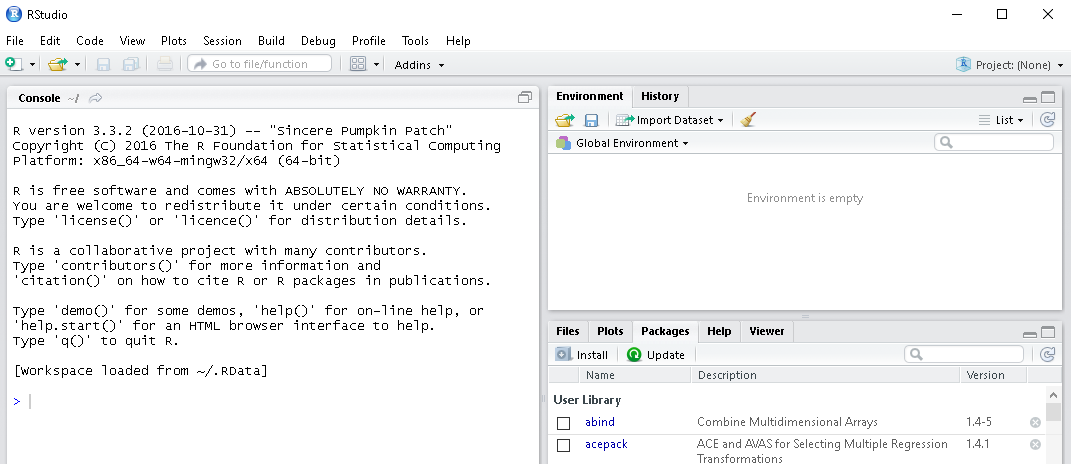
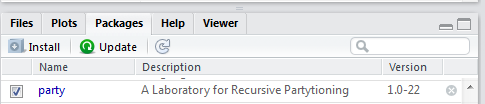


Figure 2: R Studio Interface

To run the decision tree classification method, you need to install the party package, if you have not installed it before. Enter the following command into an application console and hit enter.

install.packages("party")



Select the Packages tab at the bottom right window of an interface. Check the checkbox next to party on Figure 3 to load the package into memory.

Figure 3: Select the Packages to Load

Suppose that the cars.csv file we want to load is in the E:/Datasets folder. To set the working directory to E:/Datasets, enter the following setw command in the console window and hit the enter key. The directory path is specified in parentheses enclosed in double quotes. The slashes in the directory path should be the forward slashes.

setwd("E:/Datasets")

To verify that the working directory is set correctly, run the dir() command to display the files in the current working directory on Figure **4**.



We will use cars.csv file

Figure 4: Files in the Working Directory

Use **read.csv** command to read the cars file content into a data frame variable called cars. The first input parameter for the read.csv function is the data file name enclosed in double quotes. The second parameter, head=TRUE, specifies that the first row in the file contains the column headers. The sep parameter is the columns delimiter enclosed in double quotes. For example, sep=“,” means that the values in each data row are comma delimited.

The values delimiter

Command to Read from CSV file

cars<-read.csv(file="cars.csv", head=TRUE, sep=",")

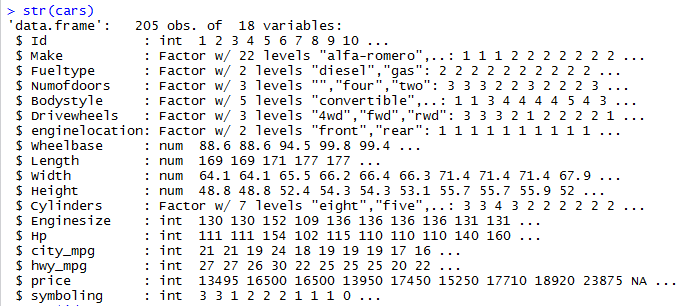
Data frame name – stores data from the first sheet in CSV file

Read the column headings from the first row

File Name

Run the str command to display the dataset structure on Figure 5. The dataset contains 205 observations/data rows and 18 variables. The first column is the list of variable names.

str(cars)



Number of Observations

Number of variables

Variable Names

Figure 5: Dataset Structure

The first variable in the dataset is a unique identifier. The unique identifiers may affect the algorithm results if they are not removed at the data pre-processing stage. In addition, the unique identifier values are irrelevant to the study. For instance, we are not going to classify the cars based on their unique identification number.

To remove the id variable, we set it equal to NULL

NULL needs to be in the upper case.

cars$Id<-NULL

Variable symbolling was read as numeric instead of factor. Run the following factor command to convert the attribute type to factor variable.

cars$symboling<-factor(cars$symboling)

To validate that the symbolling is now a factor variable, run the summary command on Figure 6 to display the variable statistics. The statistics include the list of symbolling values and the number of data rows that have each value.



Figure 6: Symbolling Statistics after Factor Function

# Running the ctree Method

## Divide Data into Training Set and Test Set

We divide the data into training set and test set. We use the training set to build the classification tree model, and we use the test set to evaluate the accuracy of a model. The training set will contain 70% of data, and the test set will contain 30% of data.

Setting the seed value enables reproducing the results when the method is rerun.

set.seed(1234)

ind <- sample(2, nrow(cars), replace = TRUE, prob = c(0.7, 0.3))

train.data <- cars[ind == 1, ]

test.data <- cars[ind == 2, ]

## Use the Training Set to Build the Model

The ctreee function builds a decision tree. It takes the formula and a dataset name as an input. A formula is an expression that contains the dependent variable followed by ~ (tilde) and independent variables. The dependent variables are delimited by + (plus sign)

For example, symboling~Make+width means that symbolling is a dependent variable, and Make and width are the independent variables.

To use all remaining variables as predictors, we may use a wildcard character . (dot) instead of listing all variable names.

Enter the first command to create the formula where symboling is a dependent variable, and all remaining variables are independent variables - character . (dot) on the right hand side of ~(tilde)

Enter the second command to build the model using the training data and the formula we just created. Store the model in a variable called cars\_ctree.

myFormula<-symboling~.

cars\_ctree <- ctree(myFormula, data = train.data)

Run the following print command to display the model stored in the cars\_ctree variable on Figure 7. The results include the number of leaf nodes, dependent and independent variables, number of observations in the training set, and variables/values used for the splits. Each terminal node is marked with a \* (star), and the tree has 7 terminal nodes.

print(cars\_ctree)

Numofdoors is the first splitting attribute. If the value is four or is missing, make is the second splitting attribute.

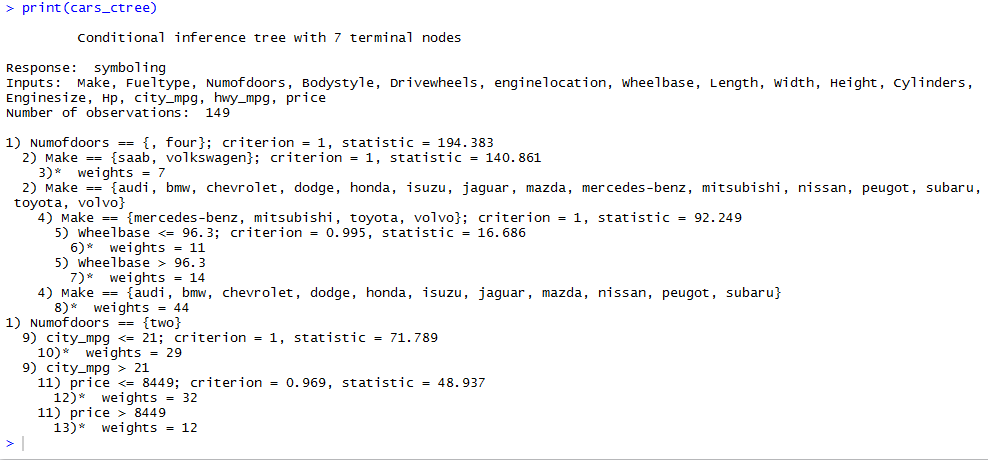
If Numofdoors is two, city\_mpg is the second splitting attribute.

If Numofdoors is two and city\_mpg<=21, we have reached the terminal node 10.

If Numofdoors is two and city\_mpg>21, price is the third splitting attribute.

If Numofdoors is two and city\_mpg>21 and price<=8449, we have reached he terminal node 12

If Numofdoors is two and city\_mpg>21 and price>8449, we have reached he terminal node 13



**City\_mpg is the second splitting attribute if numofdoors is two**

Numofdoors is the first splitting variable

**Make is the second splitting variable if numofdoors is missing or is four**

A star indicates a terminal node

Number of leaf nodes

Number of instances in the training set

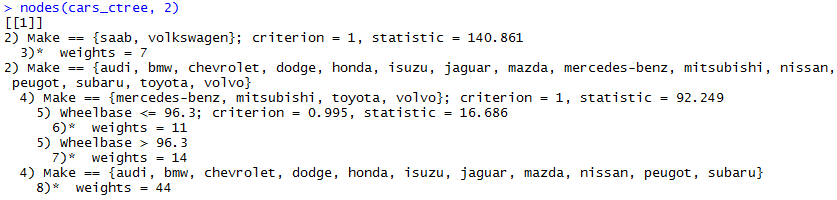
Independent Variables

Dependent Variable

Figure 7: ctree Method Results

You may use the nodes function to print the partial tree structure. The nodes function takes the model name and the starting node number as an input. Enter the following command to print the tree section on Figure 8, which starts from node 2. Note that the partial tree excludes the siblings. Node 9 and node 2 are the siblings because they share a common parent node 1. Node 9 and it’s children are excluded.

nodes(cars\_ctree, 2)



**Make is the second splitting variable if numofdoors is missing or is four**

Figure 8: Subtree Starting from Node 2

Run the plot command to graph the tree on Figure 9. The tree contains 13 nodes.

plot(cars\_ctree)

**Node 1** is the root node. The variable name specified in the node circle is a splitting variable. The value of the splitting variable determines the tree traversal. If Numofdoors=two, follow the right branch. If Numofdoors=four or if the Numofdoors value is missing, follow the left branch.

**Nodes 2 and 4** use the values of the Make attribute for splitting. Node 2 is a direct child of the root node, and node 4 is the direct child of node 2.

**Node 3** is another direct child of the root node, and city\_mog is selected for the dependent variable. If the value is less than or equal to 21, we take the left branch. If value is above 21, we take the right branch.

Nodes 3, 6, 7, 8, 10,12, and 13 are the terminal nodes or the leaf nodes. For each terminal node, the model shows the node number, number of instances in the node, and the histogram. The histogram shows the percentage instances in the node that has each symbolling value.

The dominant symbolling value becomes the terminal node class. For example, the histogram for the terminal node 3 shows that approximately 90% of instances have symboling=2. Hence, node 3 class is 2.

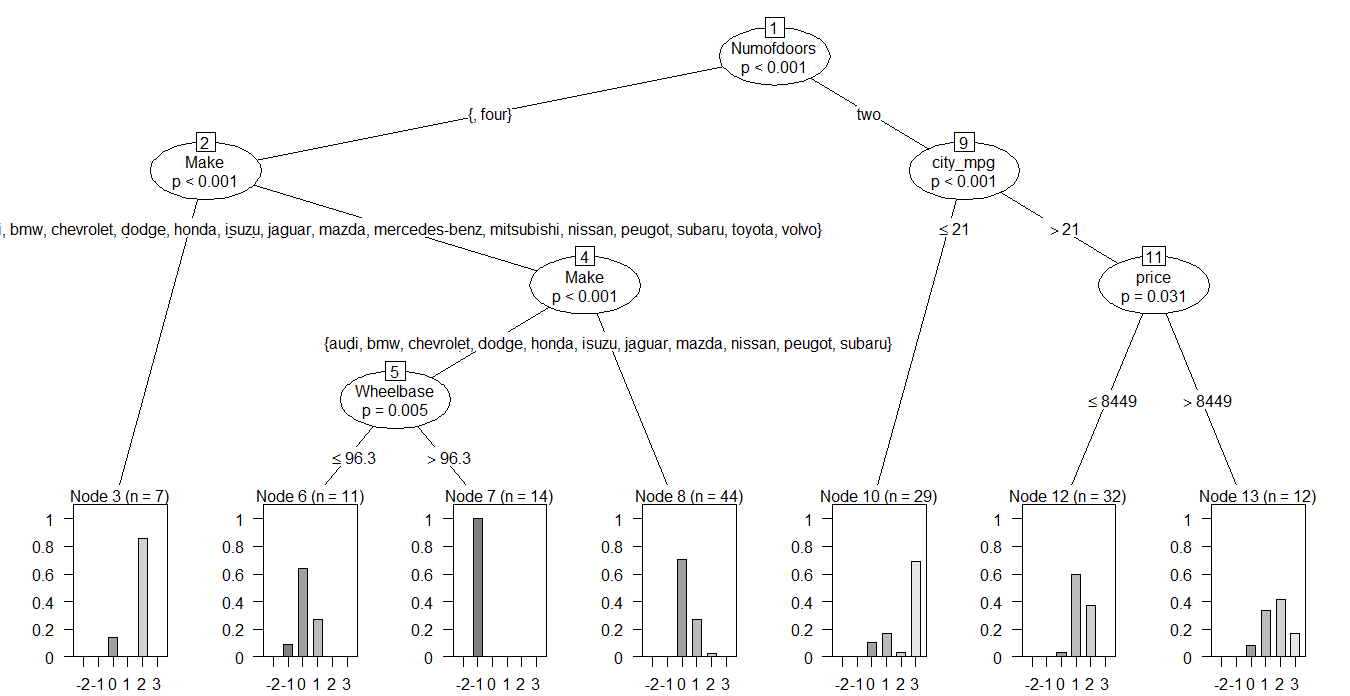


Figure 9: Tree Plot

Run the following command to plot a simpler version of the tree on Figure 10.

plot(cars\_ctree, type="simple")

The tree contains 13 notes, and 7 nodes are the terminal nodes. Instead of the histogram, each terminal node contains a vector representing the proportions of instances in the node that have each symbolling value. For example, the entries in the node 3 vector correspond to the following symbolling values.

Y= (0, 0, 0.143, 0, 0.857, 0)

Symboling Values -2 -1 0 1 2 3

The proportion of instances with symboling 0 is 0.143, and the proportion of instances with symboling 2 is 0.857. Hence, the estimated class probabilities for node 3 are

Class -2, -1, 1, 2 - 0%

Class 0 - 14.3%

Class 2 - 85.7%

Class 2 is the dominant class for node 3.

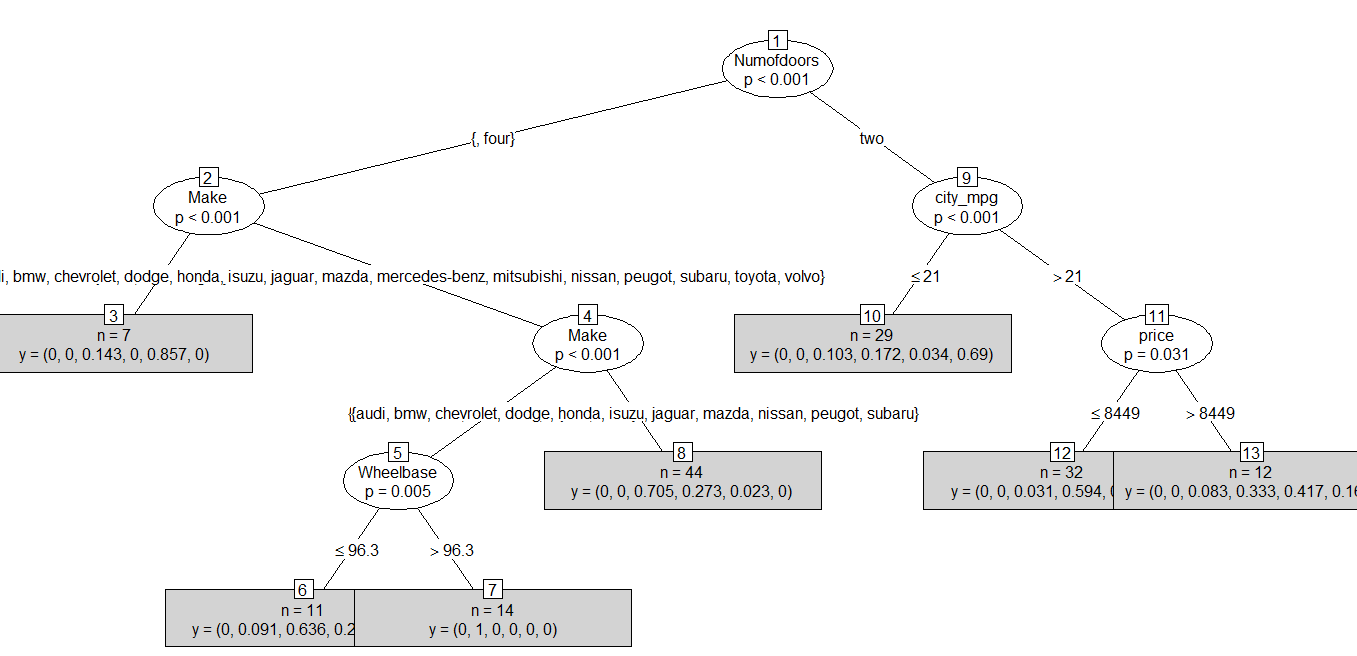


Figure 10: Simpler Tree Plot

## Check the Classification Accuracy for the Training data

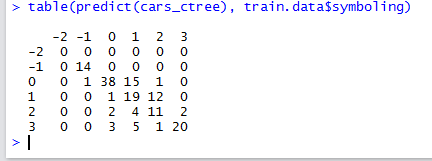
Run the following table command to build a confusion matrix for the training set on Figure 11. The **predict** command uses the model to predict the symboling class for the instances. It takes the model as an input. The table command takes the predicted class and an actual class as an input.

Predicted symboling

Actual symboling

> table(predict(cars\_ctree), train.data$symboling)

A **Confusion matrix** shows how many cars in the test data have been assigned to each class. For each matrix element, the row label is a predicted class, and the column label is an actual class. The number of correctly classified instances is the sum of numbers on diagonal from top left to bottom right. The sum of numbers outside the diagonal from top left to bottom right is the number of misclassified instances. The sum of all matrix entries is the number of instances in the training set.



The number of correctly classified instances= 0+14+38+19+11+20=102

The number of misclassified instances=15+12+2+1+1+1+4+1+2+5+3=47

The number of instances in the training set. 102+47=149

The classification accuracy is sum of numbers on diagonal/sum of all numbers=0.685 or 68.5%

Actual class

Predicted class

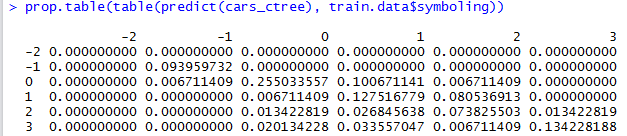
Figure 11: Confusion Matrix for Training Set

We may use a prop.table command to compute the probability for each matrix entry. The command takes the table object as an input.

prop.table(table(predict(cars\_ctree), train.data$symboling))

Figure 12 shows the table for the confusion matrix above. The column names are the actual class values, and the row names are the predicted class values. Each value in the intersection is a probability that an instance has an actual class=corresponding column name and predicted class=corresponding row name. For example, the probability that an instance has an actual class=0 and predicted class=3 is 0.02 since it’s the value at the intersection of column named 0 and row named 3.

The sum of numbers on a diagonal from upper left to lower right is the classification accuracy. The sum of all matrix entries is 1.



Actual class=0

Predicted class=3

Predicted class

Actual class

Figure 12: Prop Table

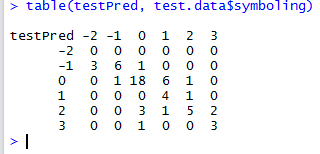
## Evaluate the Model for the Test Data

Run the following commands to evaluate the model for the test data and to build a confusion matrix. For the prediction to work, the values of the categorical variables (levels) in the test data must be the same as the values in the training data. In addition, the number of variables and variable names in the training set need to match the number of variables and the variable names in the test set.

The predict command takes the model name. The second parameter newdata indicates that the predictions will be made for the test data. The table command takes the predicted symboling and actual symboling values for the test data.

testPred <- predict(cars\_ctree, newdata = test.data)

table (testPred, test.data$symboling)



The number of correctly classified instances= 0+6+18+4+5+3=36

The number of misclassified instances=3+1+1+3+6+1+1+1+2=20

The number of instances in the training set. 36+19=56

The classification accuracy is sum of numbers on diagonal/sum of all numbers=0.642 or 64.2%

Figure 13: Confusion Matrix for Test Data

Hence, the classification accuracy for the training set is 64.2%.

Run the method on different attribute subset. Does the classification accuracy improve? What additional data pre-processing would you recommend to improve the accuracy of a model?

# Troubleshooting

**Issue** – the command to attach the party package returns the “package zoo could not be found” error on Figure 14.

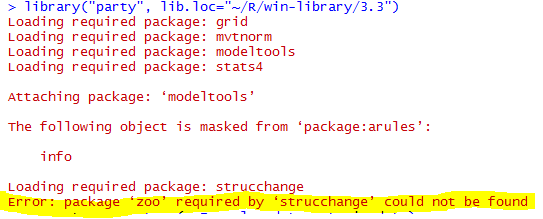


Figure 14: Possible package zoo required error

**Solution** – Run the install.packages command to install or reinstall the package zoo. Then reattempt to load the party package.

install.packages("zoo")

library("party")

An output on Figure 15 shows that package party was loaded successfully.

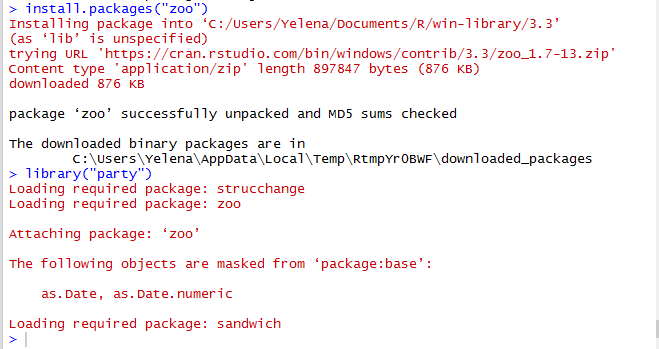


Figure 15: Package party is loaded successfully