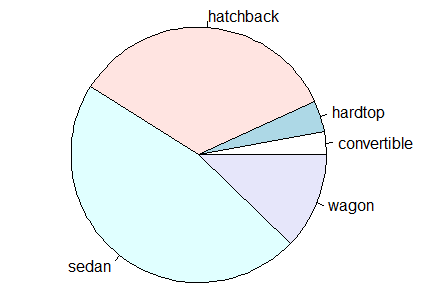
|  |
| --- |
| UMUC |
| Data Preprocessing in R |
| Week 2 Exercise |
|  |
| **DBST 667 – Data mining** |
|  |



|  |
| --- |
| In this exercise, you will use R Studio interface to load the cars data and to apply the data preprocessing filters, including variables subset selection, discretization, and data rows subset selection. You will learn about the statistics the software provides for different attribute types, four unsupervised discretization methods and the differences among them, and the attribute plots. |

# Data Preprocessing In R

[Data Preprocessing In R 1](#_Toc466320212)

[Why Preprocessing? 2](#_Toc466320213)

[Launch the Program 2](#_Toc466320214)

[Set the Working Directory 3](#_Toc466320215)

[Loading Data into R 4](#_Toc466320216)

[Exploring the Data 5](#_Toc466320217)

[Variable Filters 11](#_Toc466320218)

[Factor Function 11](#_Toc466320219)

[Discretization 12](#_Toc466320220)

[Cut Function 15](#_Toc466320221)

[Deleting a Variable 16](#_Toc466320222)

[Row filters 18](#_Toc466320223)

[Rows Range 18](#_Toc466320224)

[Handling Missing Values 19](#_Toc466320225)

[Remove Rows with Missing Values 19](#_Toc466320226)

[Display the Rows with Missing Values 20](#_Toc466320227)

[Number of Missing Values in a Data Row 21](#_Toc466320228)

[Number of Missing Values for Each Variable 21](#_Toc466320229)

[Replacing missing values with the mean 22](#_Toc466320230)

[Subset of Rows Matching an Equality Condition 23](#_Toc466320231)

[Sorting 23](#_Toc466320232)

[Saving Data to CSV File 23](#_Toc466320233)

[Data visualization 24](#_Toc466320234)

[Generic Plot function 24](#_Toc466320235)

[Copying a Plot to Clipboard 25](#_Toc466320236)

[Saving a Plot as an Image 26](#_Toc466320237)

[Saving a plot as a PDF file 27](#_Toc466320238)

[Additional Plot Examples 28](#_Toc466320239)

# Why Preprocessing?

Real-world data are usually inconsistent, missing values of interest, and has errors. Since the data inconsistency and incompleteness affects the outcome of the data mining algorithm, data preprocessing is required.

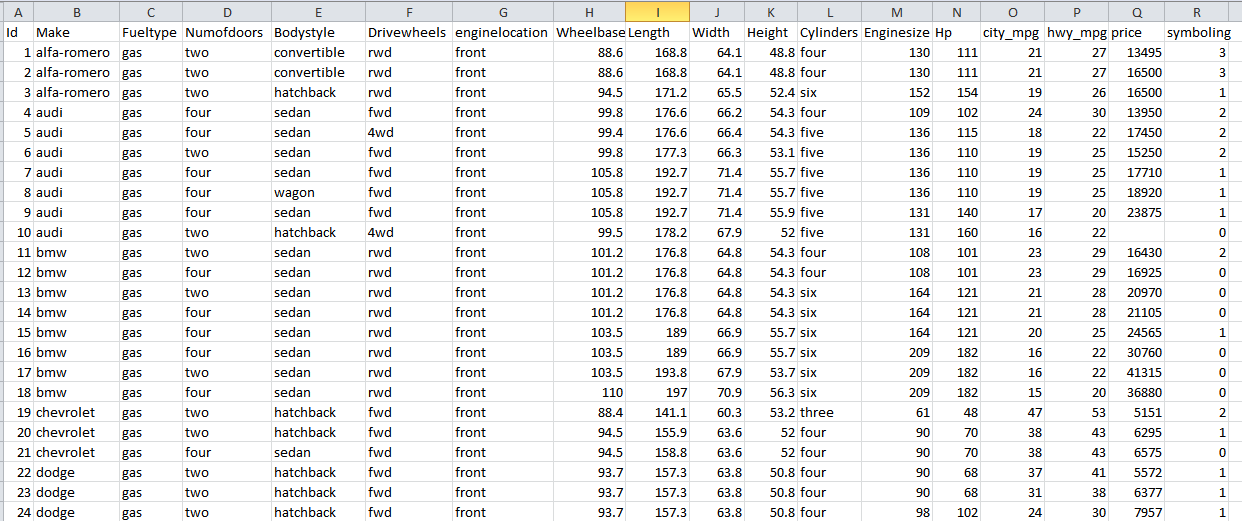
The data preprocessing operations include

* Data cleansing – filling in the missing values, removing the outliers, and correcting the values discrepancies
* Data reduction – reducing the number of records, and/or removing the irrelevant attributes
* Data discretization – converting the numeric attributes to nominal by partitioning the values into bins.

In this exercise, we will load the cars data file in R Studio application and will run the data preprocessing filters, including discretization.

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. If an attribute value is missing, the cell is left blank.

Column headers – attribute names / variables



Missing Price value

Data

(each row is a car record)

Figure 1: Partial Content of cars.csv file

# Launch the Program

Select RStudo from the start menu on Figure 2 to launch the program. You may also double click on RStudio Shortcut if you crated one.

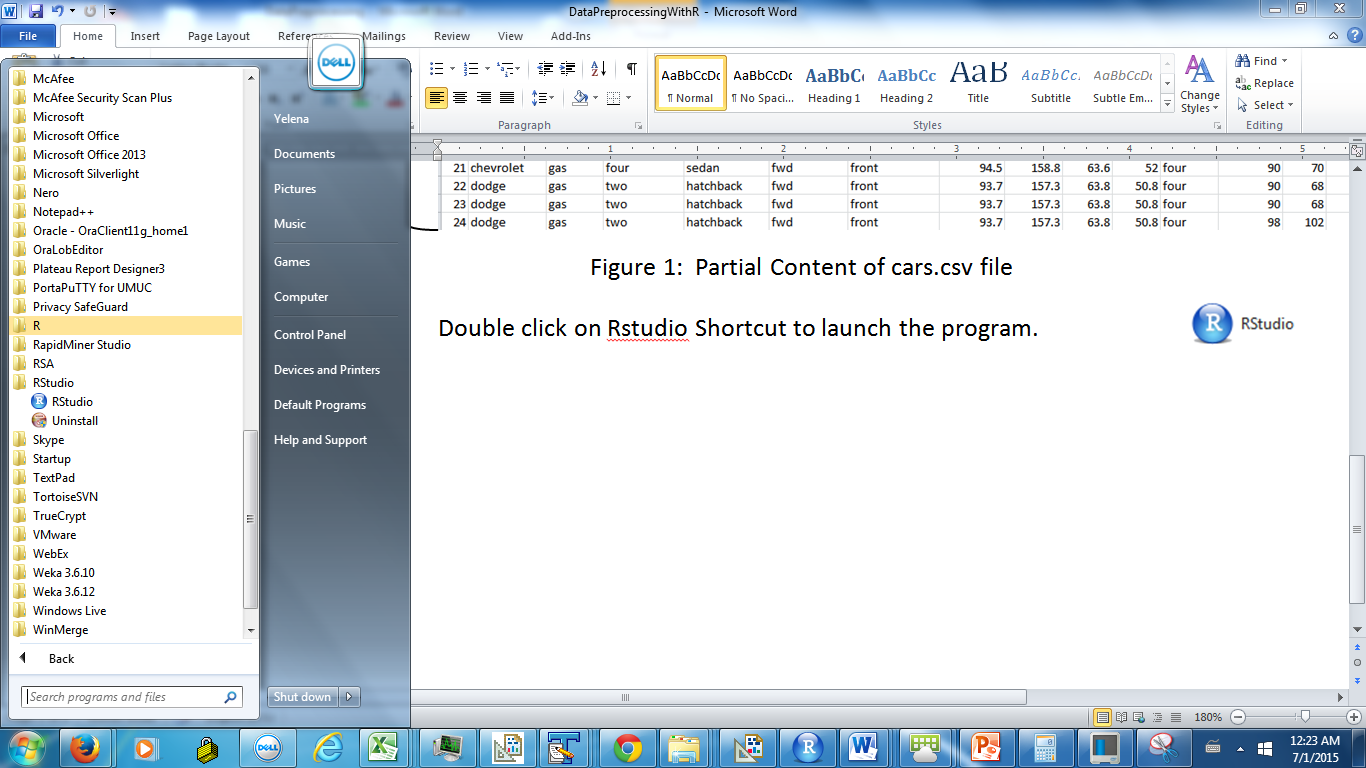
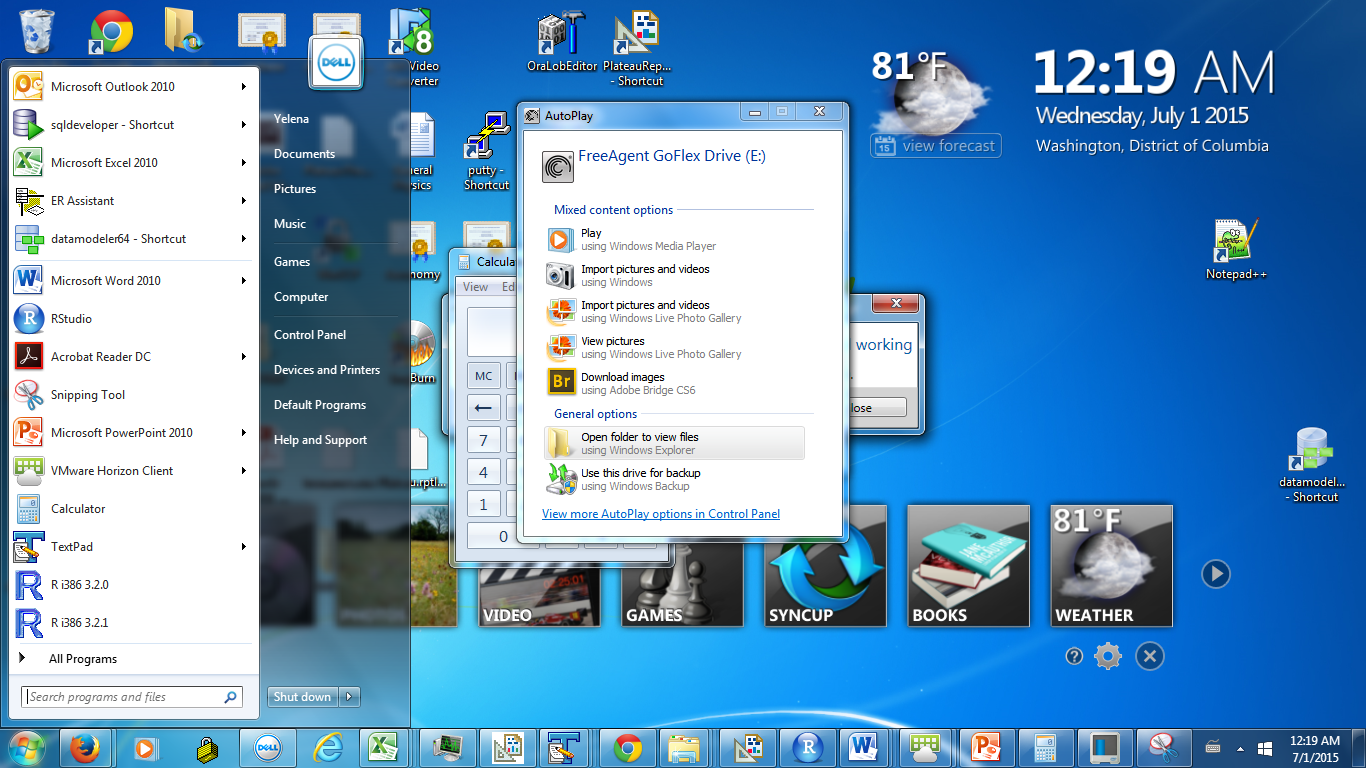
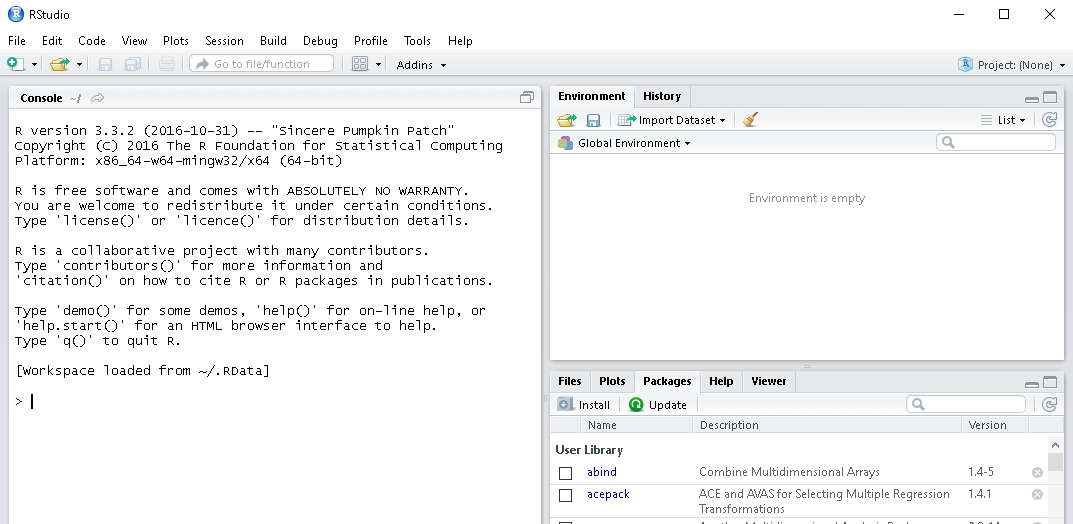


Figure 2: Select RStudio from the menu

An interface on Figure 3 will open. The left panel is a console window. You type the commands at the > prompt.

The environment tab at the top right window shows the variables and data in memory. The History tab allows viewing the commands you ran earlier. Each time you run a command, a new entry is added to the history.



Packages- software add-ons. Some packages are preinstalled, and some need to be installed.

Command Prompt

R version

The plots are displayed in the plots tab

Data and Variables in memory – the panel is empty since we just launched the program.

Files tab allows browsing the files on pc

R-studio built-in help

Session Commands History

Console- Frame where you enter the commands

Figure 3: RStudio Interface

R language is **case sensitive**. For instance, Dir and dir have a different meaning.

# Set the Working Directory

A **working directory** is a hard drive location where R looks for a file that you attempt to open. 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.

setwd("E:/Datasets")

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



Figure 4: Files in the current directory

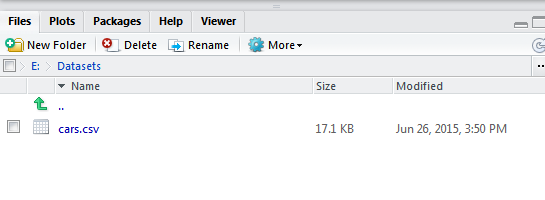
In addition, the title bar of the console window on Figure 5 displays the current working directory.



Working directory

Figure 5: Console Window Tile Bar

An alternative way to view the files in the current working directory is to select the files tab in the bottom right panel on Figure 6.



Current Working Directory

Files in the Current Working Directory

Files tab is active

Figure 6: Files Tab

# Loading Data into R

Use read.csv command to read the cars file content into a variable called cars. A variable that stores the content from the first sheet in a CSV file is called a **data frame**.

* The data frame structure resembles the matrix structure. However, the variable data types can be different.
* The data frame can also be viewed as a set of variables with the same number of values.
* The variable name cannot start with a number and cannot not contain the special characters, including !, @, $, @, +, -. And \*
* The two character operator <- is an assignment operator.

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

Assignment Operator

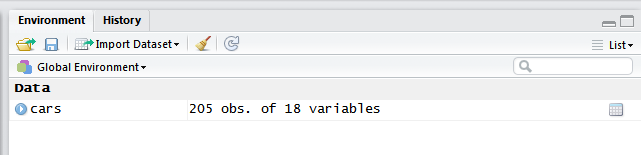
cars<-read.csv(file="cars.csv", head=TRUE, sep=",")

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

File Name

Read the column headings from the first row

Select Environment tab in the top right panel on Figure 7. Upon successful read.csv command execution, you will see cars data frame information under the data heading. The information includes the number of observations and the number of variables.



Click to view the data in a tabular format

Cars Data Frame contains 205 observations and 18 variables

Data frame name

Environment tab is active

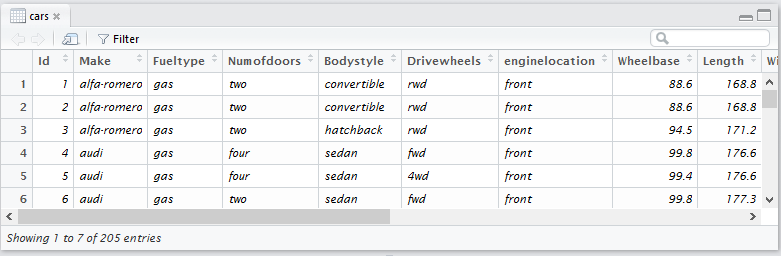
Figure 7: Cars has been added to Environment Tab

To preview the cars data in a tabular format, enter the View command at the prompt. The first letter of the command is in the upper case.

View(cars)

The first row in the preview panel shows the variable names, and the remaining rows are the data. The status bar at the bottom of the panel shows the number of data rows in the dataset. Make sure that the number matches the number of rows in the .csv file.

The arrows next to the column headings allow sorting the data rows the corresponding variable values.



Number of rows in the dataset

Use the arrows for sorting data

Data frame name

Figure 8: Data Preview

# Exploring the Data

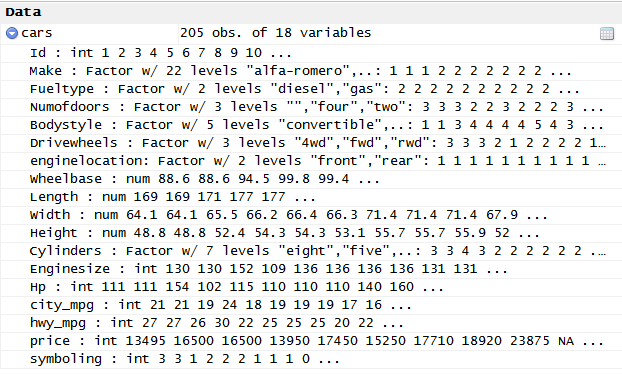
Click on a blue arrow to expand the data frame properties on Figure 9. In addition to the number of observations and the number of variables, you will see the properties for each variable, including the variable name and variable type. The variable types are:

**int** – The values of a variable are continuous integers

**num** - The vales of a variable are the real numbers

**Factor** – The values of a variable are the discrete pre-defined labels. For each factor variable, the properties include the pre-defined discrete values, called levels. For example, diesel and gas are the fueltype levels.

The data type of a variable determines what methods and operations we may use. For example, we cannot add or multiply the categorical variables. On the other hand, we cannot use some methods such as association rules for continuous numeric variables.



Number of variables

Number of observations

Variable data type

Variable name

Figure 9: Cars Data in Environment Panel

Enter the following **names** command to display the variable names we just loaded from the cars.csv file. The command takes the data frame name as an input parameter. The name is not enclosed in double quotes. Make sure that the number of variables and the variables names on Figure 10 match the number of populated columns and the column headings in the csv file. (Note that if the names contained spaces, the spaces would have been changed to periods.)

names(cars)

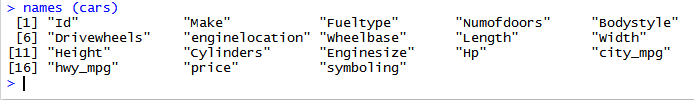
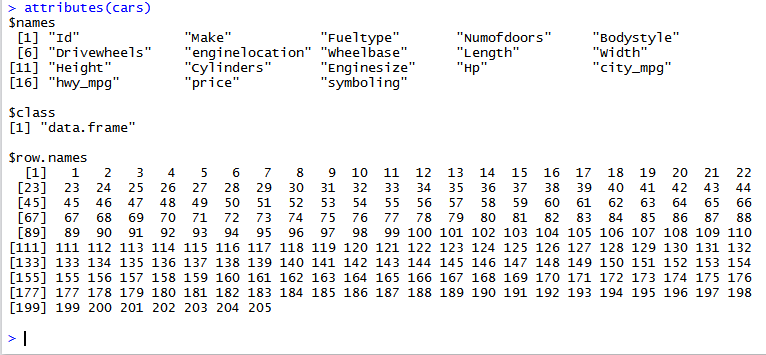


Figure 10: Attribute names

Each data row in the data frame is named with the corresponding value in the first column. To display the row labels in addition to the variable names on Figure 11, enter the attributes command. The command takes the data frame name as an input.

attributes(cars)



Data row names

Data Frame name

Variable names

Figure 11: Variable names and row names

To display the size of the data frame, run the **dim** command. The command takes the name of the data frame as an input. The first number in the output on Figure 12 is the number of data rows, and the second number is the number of variables.

dim (cars)

18 Variables



205 Data Rows

Figure 12: Dim Command output

To display just the number of data rows, run the following **nrow** command. The command takes the data frame name as an input.

nrow(cars)

The output on Figure 13 shows that the cars data contains 205 rows.



Figure 13: Number of Rows

The following **dim** command also returns the number of rows. 1 means the first dimension of the data frame, or the number of rows.

dim(cars)[1]

To display the number of variables, run the following **ncol** command. The command takes the data frame name as an input.

ncol(cars)

The output on Figure 14 shows that the cars data has 18 variables.



Figure 14: Number of Variables

The following **length** and **dim** command also return the number of variables. The [2] means the second dimension of the data frame, or number of columns.

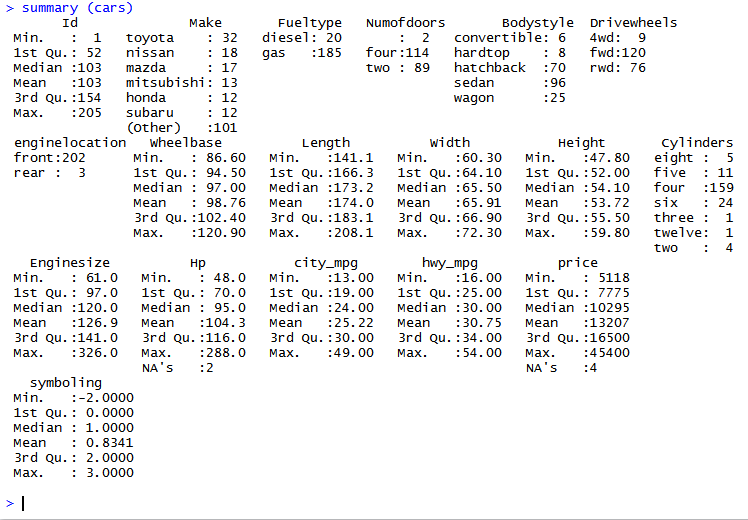
length(cars)

dim(cars)[2]

Run the **summary** command to display the descriptive statistics for all variables or for a single variable. To display the statistics for all variables, specify the data frame name as a summary command input parameter.

summary (cars)

Figure 15 shows the output from the summary command. An output includes a table with statistics for each variable in the data frame.



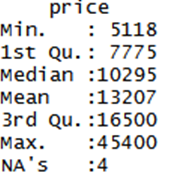
**NA stands for missing values.**

Factor Variable

Numeric Variable

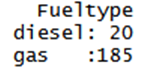
Figure 15: Descriptive Statistics for All Variables

The statistics for continuous numeric variables are the minimum value, maximum value, 1st and 3rd quartile, median, and mean. The statistics also include the number of instances with missing attribute value, if there are any.



For example, variable price is because it does not have a pre-defined set of discrete values. The values of a price are the continuous real numbers. The NA’s :4 indicates that 4 car instances are missing a price value.

A factor variable is a categorical variable with pre-defined discrete values called **levels**. The factor variable statistics are the levels and the number of data rows that have value at each level. For example, the levels of the Fueltype variable are diesel and gas.



185 cars have a gas fuel type

20 cars have a diesel fuel type

Levels

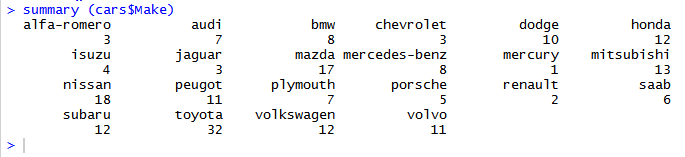
You may use summary command to display the descriptive statistics for a single variable. Append the $ and the variale name to the data frame name. For example, type the following command to display the descriptive statistics for the Make attribute.

summary (cars$Make)

Since Make is a factor variable, the statistics are the valid make are the levels and the number of values at each level.

An individual variable name

Data Frame name



Each car make value is a level

Figure 16: Descriptive statistics for Make attributes

You may also use the following lapply command to display the descriptive statistics for all variables. The command loops through the cars data frame variable and displays the statistics for each variable. Figure 17 shows the partial output of the lapply command.

The first input parameter is the data frame name, and the second parameter is the function to run for each variable.

lapply(cars, summary)

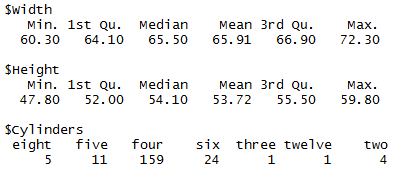


Figure 17: Partial Output from lapply command

To display the minimum and maximum value of the numeric variable, use the range function. The function takes the variable as an input – data frame name followed by $ followed by variable name.

range(cars$Width)

Figure 18 shows that the lowest car width value is 60.3 and the highest car width value is 72.3.



Lowest value

Highest value

Figure 18: Car Width Range

Notice that the output from the summary command excludes the standard deviation statistics. To display the standard deviation for numeric variable, run the sd command. The command takes variable name as an input.

sd(cars$Width)

An output on Figure 19 is the standard deviation of the car width variable.



Figure 19: Car Width Standard Deviation

What happens when we run the mean command on a variable with missing values? Run the following command to display the mean of a price variable.

mean(cars$price)

The output on Figure 20 shows that the mean value of a variable is unknown if at least one value is missing.



Figure 20: Mean of a Price Variable with Missing Values

The optional second parameter for a mean function enables ignoring the missing values in the mean computation. We set the parameter na.rm=TRUE to ignore the missing values. When the parameter is omitted, the default value is FALSE.

mean(cars$price, na.rm=TRUE)

The output on Figure 21 is the mean car price.

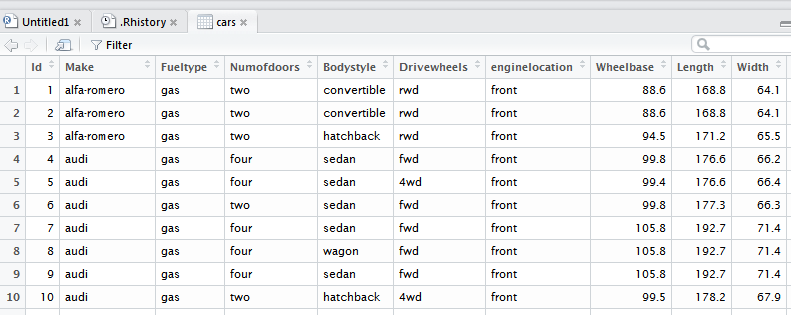


Figure 21: Mean of a Price Variable with Missing Values Ignored

To preview the cars data in a tabular format, enter the View command at the prompt. The first letter of the command is in the upper case.

View(cars)

Figure 22 shows the partial cars data in the tabular view. Each column corresponds to a variable. The first row is a list of the variable names. The remaining rows are the data.



Click on arrows next to variable name to sort the data.

Data Rows

Headers – Variable names

Figure 22: Cars Data in a Tabular View

# Variable Filters

The variable filters add variables, remove variables, and change variable values. The number of data rows remains the same.

## Factor Function

The factor function converts the variable with discrete numeric values into a categorical factor variable.

The last variable symbolling has the categorical values -2, -1, 0, 1, 2, 3. However, the last line on Figure 9 shows that the variable is numeric. In addition, an output from the summary command on Figure 23 includes the descriptive statistics for numeric variables (min, median, mean, etc.)

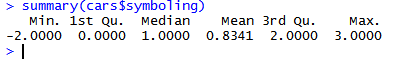


Figure 23: Symboling attribute statistics before factor

To convert the variable to a factor datatype, we run a factor function. The function take a data frame variable in a data frame as a parameter. <- is an assignment operator.

cars$symboling<-factor(cars$symboling)

To validate if the variable has been converted from numeric to factor, we run the summary command on Figure 24. The statistics have changed from min, median, mean, etc. to the levels and the number of data rows with the value at each level.



27 cars have symboling level 3

The symboling values are the Levels

Figure 24: Symboling variable statistics after factor

## Discretization

**Discretization** is the process of converting numeric variables into categorical variables (factors) by dividing the numeric values into distinct ranges. Some algorithms such as association rules, covered later in the course, work only with the factor variables.

In some textbooks and online documentation, the authors may refer to discretization as binning or recoding.

**Unsupervised discretization** does not consider the class of a data instance. The dependent variable can be either numeric, or integer, or factor.

To run the unsupervised discretization, you need to install the discretization package if you have not installed it before. Enter the following command into an application console and hit enter. Figure 25 shows the successful package installation confirmation.

install.packages("discretization")

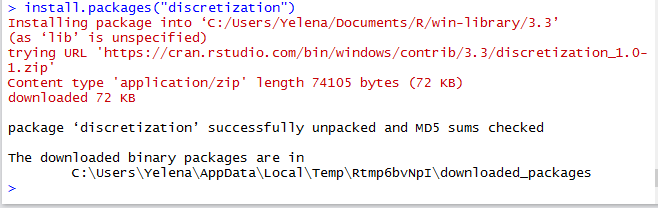


Figure 25: Discretization Package Installation confirmation.

You also need to install the arules package if you have not installed it before.

install.packages("arules")

Run the load command to call the package. The command needs to be run each session when you need to use the package. Load discretization and arules package.

library (discretization)

library (arules)

**Equal Interval width** is a default unsupervised discretization method. The values partitioned into intervals of equal width. This approach is sensitive to the extremely large and/or extremely small values called the outliers. In addition, this approach does guarantee that initial distribution of values remains.

We will run the equal interval width method on city\_mpg variable to partition the values into six ranges.

Figure 26 shows the city\_mpg variable statistics before running the discretization. The statistics are minimum value, median, mean, etc.



Figure 26: City\_mpg Statistics before Discretization

Enter the following command into the console and hit the enter key. The first parameter in parentheses specifies the variable we want to discretize. The second parameter is the discretization method, which is interval. The third parameter is the number of ranges.

Discretization Type

Variable Name

Number of categories

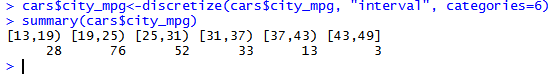
cars$city\_mpg<-discretize(cars$city\_mpg, "interval", categories=6)

An output from the summary command on Figure 27 shows the city\_mpg variable statistics after discretization. The value ranges (levels) and the number of cars that have a city\_mpg value in each range.

For example, 28 cars have a city\_mpg between 13 and 19. 76 cars have a city\_mpg between 19 and 25. The value ranges have approximately the same width.

The round bracket indicates that the boundary point does not belong to an interval. The square bracket indicates that the boundary point belongs to an interval.

summary(cars$city\_mpg)



Value ranges = levels

Figure 27: city\_mpg Variable After Discretization

**Equal Frequency Method** – The interval length is based on the variable values distribution. Approximately the same number of data instances should have a value in the same data range.

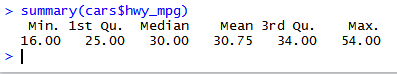


Figure 28: hwy\_mpg Variable Before Discretization

Number of categories

Discretization Type

Variable Name

cars$hwy\_mpg<-discretize(cars$hwy\_mpg, "frequency", categories=6)

Figure 29 shows the value intervals (levels) after the discretization. For each interval, the round bracket indicates that the boundary point does not belong to an interval. The square bracket indicates that the boundary point belongs to an interval.

summary(cars$hwy\_mpg)

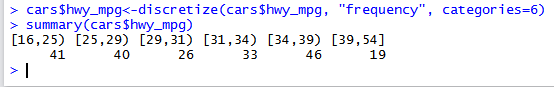


Figure 29: hwy\_mpg Variable After Discretization

**K-means clustering** discretization method is also known as a univariate k-means clustering method. The algorithm randomly selects the center points for each partition. Each value within partition is closer to the partition’s center point than to the center points in other partition. This discretization approach attempts to preserve the original values distribution. (The k-means clustering method will be covered later in the course).

Figure 31 shows the Enginesize attribute statistics before running the discretization.



Figure 30: Enginesize Variable before Discretization

Type the following command and hit enter. The first parameter is a variable name. The second parameter is a discretization method, which is cluster. The third parameter is the number of value intervals.

Variable Name

Discretization Type

Number of categories

cars$Enginesize<-discretize(cars$Enginesize, "cluster", categories=6)

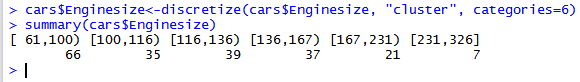


Figure 31: Enginesize Variable after Discretization

**User Specified Method** – a user specifies the boundary points for each interval.

Suppose that we want to divide the wheelbase attribute values into the following ranges:

Wheelbase below **90**

Wheelbase between **90** and ***100***

Wheelbase between ***100*** and *110*

Wheelbase above *110*

The boundary points are negative infinity, 90, 100, 110, and positive infinity.

Figure 32 shows the Wheelbase variable statistics before running the discretization filter.

summary(cars$Wheelbase)

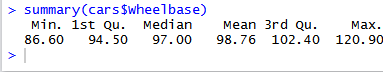


Figure 32: Wheelbase Variable before Discretization

Enter the following command and hit the enter key. The first parameter is an attribute name. The second parameter is a discretization type, which is fixed. The third parameter is a vector of the boundary points. –Inf stands for the negative infinity, and inf stands for positive infinity.

cars$Wheelbase<-discretize(cars$Wheelbase, "fixed", categories=c(-Inf, 90, 100, 110, Inf))

Interval Boundary Points

summary(cars$Wheelbase)

Figure 33 shows the Wheelbase variable statistics after applying the discretization filter. The interval boundary points match the point specified boundaries in the input vector. The round bracket indicates that the boundary point does not belong to an interval. The square bracket indicates that the boundary point belongs to an interval.

The most dominant value range for the wheelbase is between 90 and 100. Only 8 data rows have wheelbase below 90.

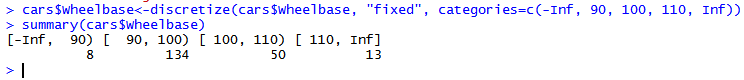


Figure 33: Wheelbase Variable after Discretization

## Cut Function

Cut function is an alternative approach to perform the unsupervised discretization with equal width binning. The function partitions the variable values into the specified equal width categories. The function takes a variable name and a number of categories input parameters.

Figure 34 shows the Length variable statistics before and after running the cut function. The statistics before running the cut function include minimum, median, mean, etc. The statistics after running the cut functions are 4 value ranges and the number of data rows with the Length value in each range.

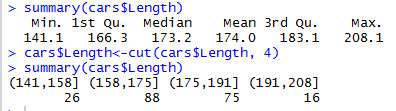
summary(cars$Length)

Number of ranges

Variable name

cars$Length<-cut(cars$Length, 4)

summary(cars$Length)

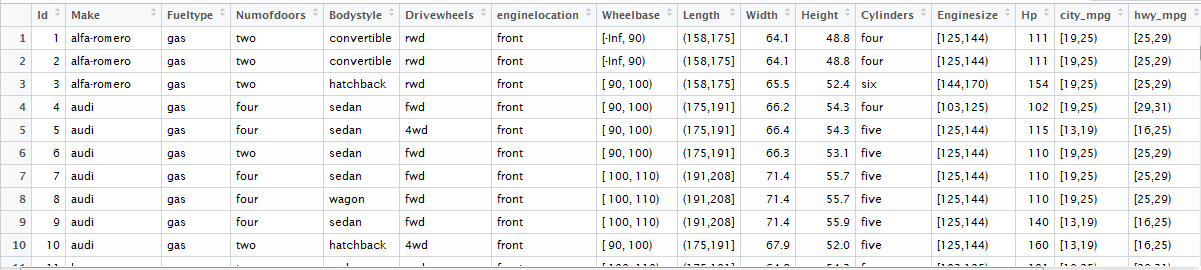


Length statistics before running cut function

Length statistics after running cut function

Figure 34: Cut Function Output

Figure 35 is a tabular view of the partial cars data after applying a discretization. The Wheelbase. Length, Enginesize, city\_mpg, and hwy\_mpg values are the interval ranges.



The value are discrete value ranges

Figure 35: Discretized Values – Tabular View

## Deleting a Variable

The id variable uniquely identifies each data row. Since unique identifiers may affect the data mining method performance, we may choose to drop them.

Enter the following command to create a copy of the cars data frame with the id variable removed. A comma after an opening bracket indicates that we are removing a column. The number following the minus sign is a position of the variable to remove.

newcars<-cars[, -1]

The newcars data frame has been added to the Environment tab in the upper right panel on Figure 36. The number of variables changed from 18 to 17. The number of observations remains unchanged.

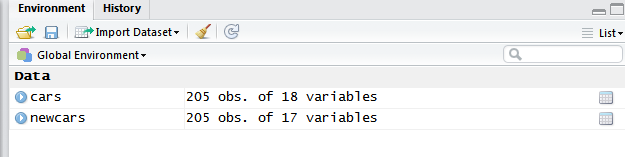


Figure 36: newcars data frame

Run the View command to preview the newcars in a tabular format on Figure 37. ID variable has been removed, and Make is the first variable.

View(newcars)

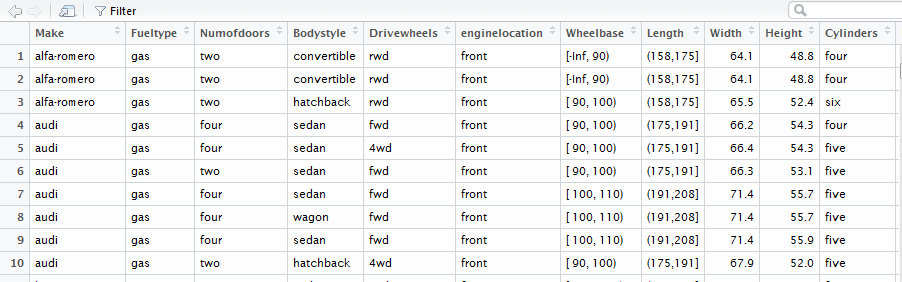


Figure 37: ID has Been Removed

To remove more than one variable in a single command, we specify the variable positions as a vector. Variables ID, fueltype, and numogdoors have position 1, 3, and 4 respectfully.

To specify the positions of a variables to remove, we need to create a vector with numeric values.

Function c(1, 3, 4) creates a numeric vector with the values 1, 3, 4. The minus sign preceding the vector means that the specified columns are removed.

newcars<-cars [, - c(1, 3, 4)]

Variables ID, fueltype, and numofdoors have been removed. Make variable has been moved to the first position. Body style has been moved to the second, etc.

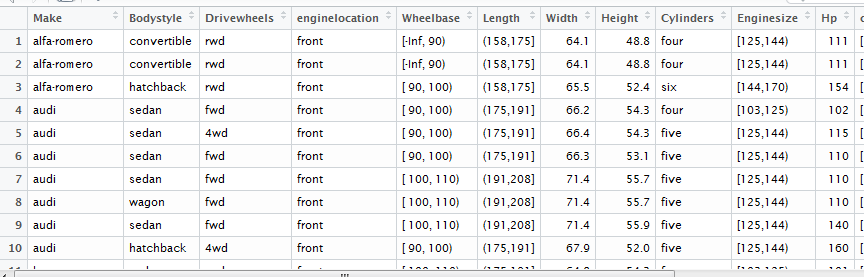


Figure 38: ID, fueltype, and numofdoors Removed

Instead of specifying the variables to remove, you may specify the variables to keep. When the minus sign preceding the vector is omitted, the variables at the positions specified in the vector are kept, and the remaining variables are removed.

Suppose that we want to keep Id, Make, Fueltype, Numofdoors, and Bodystyle. The variables are at the positions 1, 2, 3, 4, and 5 respectfully. The command takes a vector c(1, 2, 3, 4, 5) as an input.

newcars<-cars [, c(1, 2, 3, 4, 5)]

Figure 39 shows the tabular view of the partial newcars data frame. The variables in the data frame are ID, Make, Fueltype, Numofdoors, and Bodustyle.

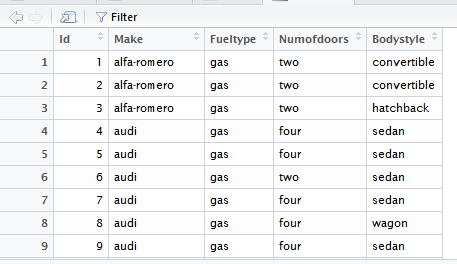


Figure 39: Only Specified Variables Remained

The following command is an alternative approach to keep the first 5 attributes. The numbers in the square bracket specify the variable positions range.

newcars<-cars[1:5]

Another approach to remove the column is to set it to NULL. The first command copies all data from cars data frame into newcars. The second command removes the Id column.

newcars<-cars

NULL needs to be in the upper case.

newcars$Id<-NULL

# Row filters

The row filters are used to add rows, remove rows, and process rows. The number of variables remain the same.

## Rows Range

To select the subset of a rows, specify the data frame followed by the square brackets. The first parameter in the brackets is the row numbers range. Missing number after a comma means that all variables are included.

newcars<-cars[1:10,]

The tabular view on Figure 40 shows that the newcars contains the first 10 rows and all variables from the cars data frame.

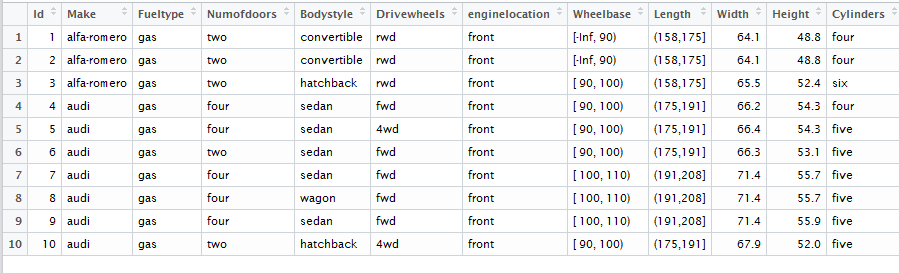


Figure 40: First 10 Rows

## Handling Missing Values

### Remove Rows with Missing Values

We may create a copy of the cars data frame with the observations missing a value for at least one variable removed. We call the data frame without missing values newcars.

**Method 1 – Use a subset command.** The command takes the data frame name and the logical condition as an input. The rows that meet the condition are part of the subset. The complete.cases condition is true when the data row has a value for each attribute.

newcars<-subset(cars, complete.cases(cars))

**Method 2 – Use square brackets.** Instead of using a subset function, we may specify the row matching condition in the square brackets. If we do not specify the columns after the comma inside the brackets, the subset will include all columns.

newcars<-cars[complete.cases(cars),]

**Method 3 – Use na.omit command.** The command na.omit also removes the data rows with missing values. The command takes the data frame name as an input.

newcars<-na.omit(cars)

Figure 41 shows the cars and newcars environmental variables. Cars contains 205 observations, and newcars contains 199 observations. Hence, at least 6 observations in the cars dataset are missing a value for at least one variable.

Cars and newcars have the same number of variables.

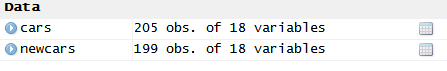
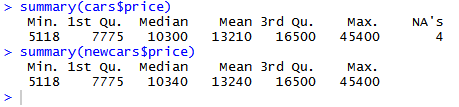


Figure 41: Number of Observations has Changed

We may also compare the statics of the individual variables. For example, an output from the summary command at the top of Figure 42 shows that 4 observations in the cars data frame are missing a price value (NA’s 4). The summary command at the bottom of the figure, shows that all observation in newcars data frame have a value. (NA’s column is omitted)



Prices has 4 missing values.

Price does not have missing values.

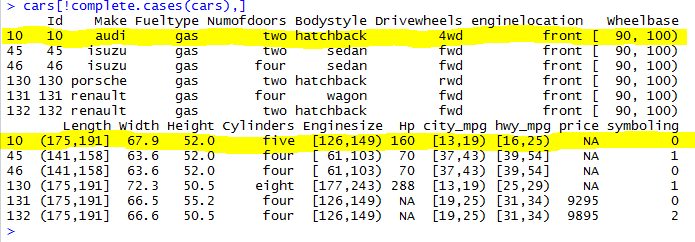
Figure 42: Price Variable Statistics Comparison

### Display the Rows with Missing Values

The following command returns the data rows missing a value for one or more variables. The ! is a logical operator not. The rows selection condition is incomplete cases. Since we did not specified the columns after a comma inside the brackets, the output on Figure 43 includes all columns.

cars[!complete.cases(cars),]

Since the variables do not fit on a single output line, the output has 2 lines for each data row. The highlighted lines on Figure 43 correspond to the same data row.



**Missing values**

Figure 43: Rows with Missing Values

If the dataset is large, we might not be interested in previewing the data rows with missing values. Instead, we could display how many data rows are missing a value for at least one variable.

The nrow function returns the rows count. The function takes a data frame or a data frame subset as an input. Instead of specifying the data frame name, we may specify the command that generates the data frame or data frame subset.

Hence, the output of the cars[!complete.cases(cars),] command discussed above is an input for nrow command.

nrow(cars[!complete.cases(cars),])

The output on Figure 44 shows that 6 data rows are missing at least one value.



Figure 44: Number of Rows with Missing Values

### Number of Missing Values in a Data Row

Whenwe remove the data rows with missing values, we risk losing data. An alternative is to fill in the missing values with the most common value for the categorical variables and with the mean value for numeric variables. However, this approach might not be the best option if the values are missing for most of the variables.

Hence, it could be useful to **check the number of missing values in each data row**. An apply function loops through the data rows one at a time, and applies the specified function to each row. The first parameter is the data frame name. To loop through rows, we specify 1 for the second parameter.

The third parameter is the function to apply. We may use the built in function, or we may define an inline function. Keep in mind that unlike regular functions, an inline function is temporary, and it is available only within the current command.

For each value in a row, is.na returns true if the value is missing and false if the value is not missing. The sum counts the number of true values returned for a row. For instance, if the data row is missing 2 values, the sum will return 2.

Is.na () is a function we use to check if the value is missing.

Data frame

Loop through rows

Inline function

apply(cars, 1, function (cars) sum(is.na(cars)))

Figure 45 shows that the first 9 rows do not have missing values. The 10th row has one missing value, etc.

10th row

First 9 rows

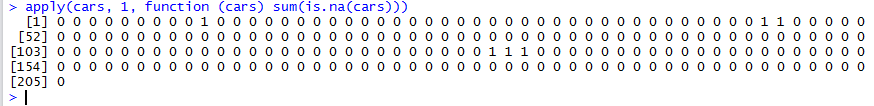


Figure 45: Number of Missing Values in Each Data Row

### Number of Missing Values for Each Variable

Removing rows with missing values and/or feeling in the missing values might not be the best approach when a variable is missing values for most of the rows. It could be useful to check how many data rows are missing a value for each variable.

We use an apply function above and change the value of the second input parameter to 2. In another words, an apply function loops through the variables instead of data rows.

apply(cars, 2, function (cars) sum(is.na(cars)))

The output on Figure 46 shows the number of missing values for each variable. For example, four data rows are missing a price value. Two data rows are missing an HP value. No data rows are missing Make value.

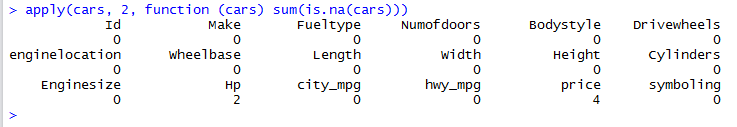


Figure 46: Number of Missing Values for Each Variable

### Replacing missing values with the mean

Since only two data rows are missing the HP value and only four data rows are missing the price, we can replace the missing values with the variable mean value.

The following command finds the rows with missing values for Hp variable and replaces the missing values with the attribute mean. In general, is.na returns true when the input value is missing. Hence, function is.na on the left hand side returns the row numbers where the Hp value is missing.

The mean function on the right hand side takes the dataset column name as a first input. The second input na.rm=TRUE means to ignore the missing values in the mean value computation.

cars$Hp[is.na(cars$Hp)]<-mean(cars$Hp, na.rm=TRUE)

Then run the following command to replace the missing price values with the price mean.

cars$price[is.na(cars$price)]<-mean(cars$price, na.rm=TRUE)

To validate that all missing values have been replace, we rerun the command to check the number of missing values for each variable.

apply(cars, 2, function (cars) sum(is.na(cars)))

The output on Figure 47 shows that all variables do not have missing values since all counts are 0.

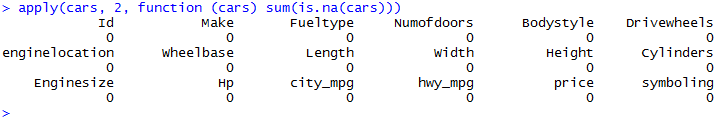


Figure 47: No More Missing Values

## Subset of Rows Matching an Equality Condition

We can take a subset of data that matches the specified equality condition. For example, the following command takes the data rows where syboling is -2. The first input parameter is the dataset name, and the second input parameter is a condition. **The double equals sign “==” is used for testing the equality**.

newcars<-subset.data.frame(cars, cars$symboling=="-2")

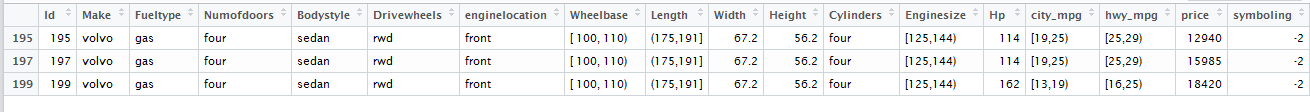


Figure 48: Tabular View – Observations with symboling =2

The following command is an alternative approach to take a subset of data rows where symboling is -2. The data frame name cars is followed by the square brackets. Inside the brackets, we specify the row selection condition followed by the comma. After the comma, we specify the variables we want to select. When the variables are unspecified, all variables are selected.

newcars<-cars [cars$symboling=="-2", ]

## Sorting

We may sort the data frame rows by values of the specified variable. The following command creates a new data frame called cars\_sorted , which is the copy of the cars data frame sorted by price ascending. We specify the name of the data frame followed by brackets. The first number in the brackets specifies the rows to select. The nested function orders(cars$price) returns an array of data row indices ordered by price value. Since the columns specification after a comma is omitted, the cars\_sorted data frame will contain all columns from cars data frame.

cars\_sorted<-cars[order(cars$price), ]

To validate that the cars\_sorted is ordered by the price descending, we run the following head command to preview the first 6 car prices on Figure 49.

By default, the head command displays the first rows. When the variable name is specified, the command returns only the values of the specified variable.

head(cars\_sorted$price)



Figure 49: First 6 Sorted Car Prices

# Saving Data to CSV File

To save the pre-processed version of the CSV file to the working directory on your hard drive, use write.csv command. The first parameter is the data frame name. The second parameter is the file name. The third parameter specifies not to add the column with the row names.

write.csv(cars, file="CarsD.csv", row.names=FALSE)

To validate if the file was saved, run the dir() command to list the files in the working directory. Figure 50 shows that the file CarsD.csv was saved into the working directory.

dir()



Figure 50: CarsD.csv File in in the Working Directory

Locate the file on a hard drive to preview the content. Figure 1 shows the partial content of the file. The first line is the variable names, and the remaining lines are the data. The values of Wheelbase, enginesize, city\_mpg, and hwy\_mpg variables are the values intervals.

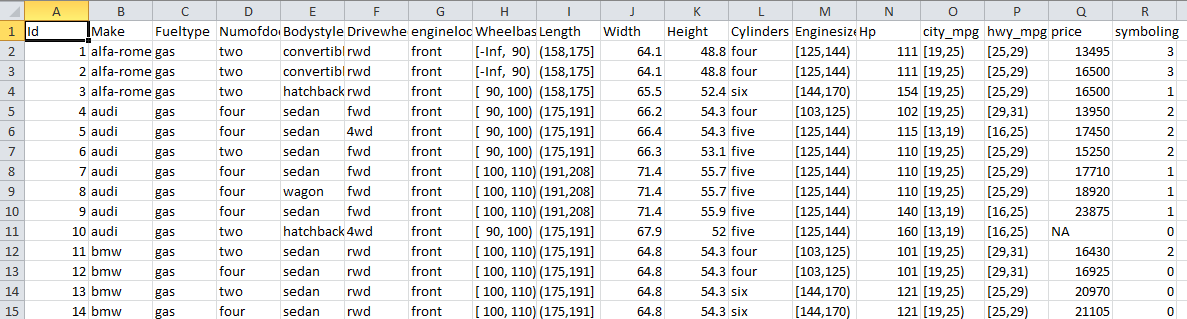


Figure 51: CarsD.csv File Partial Content

# Data visualization

## Generic Plot function

Plots facilitate an understanding of the data, including the central tendency and skewness.

The X-axis coordinates are the symbolling values. The values of the Y-axis are the number of observations that have each value. The chart indicates that 0 and 1 are the dominant symbolyng values because the bars are the highest at X=0 and at X=1.

Type h=vertical lines plot that looks similar to a histogram.

plot(table(cars$symboling), type="h", col="dark green")



Delete the current plot

Refresh the plot

Move back and forth between constructed plots

Click Zoom to open the plot in a new window

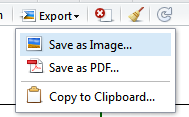
Delete all plots

Export menu allows copying the chart to the clipboard, and saving the chart as a PDF or as an image file.

Figure 52: Vertical Lines Plot

## Copying a Plot to Clipboard

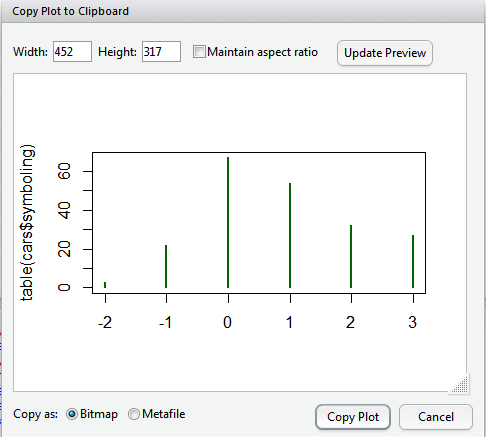
Click on and arrow to expand the Export menu on Figure 53. Select Copy to Clipboard.



Select Copy to Clipboard

Figure 53: Select Copy to Clipboard from Export Menu

The Copy to Plot to Clipboard window on Figure 54 will open. You may adjust the width and height of the plot by changing the Width and Height value. Click on copy plot button. The window will close. Return to the Microsoft Word document and select paste from an edit menu to paste the plot.



Click Copy plot to continue

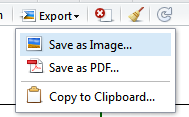
Update the Image preview after the size readjustment

You may adjust the width and Height.

Figure 54: Copy Plot to Clipboard Window

## Saving a Plot as an Image

Click on and arrow to expand the Export menu on Figure 55. Select Save as image.

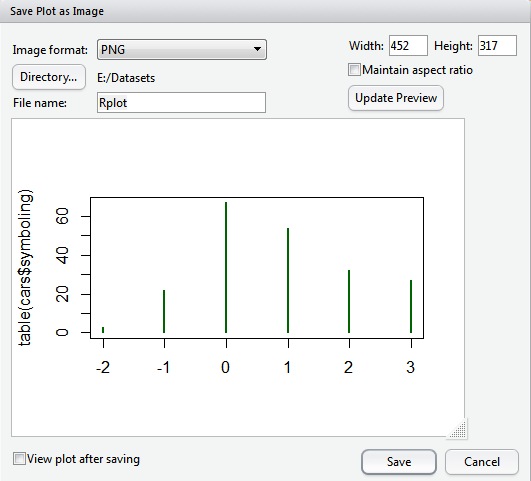


Select Save as Image

Figure 55: Select Save as Image from Export Menu

The save plot as Image window on Figure 56 will open. The drop down at the top left allows selecting an image format. In addition to PNG, the supported formats include GPEG, BMP, and SVG.

Click on Directory button to select the folder where you want to save an image. By default, the chart will be saved to the working directory specified above.

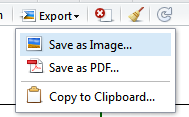


1. Readjust the image size if necessary
2. Click to Update Preview to preview the readjustments.
3. Click Save to continue
4. Choose the image format
5. Browse to the folder here you want to save the image (the default path is a working directory.
6. Enter the name or keep the default name

Figure 56: Save Plot as Image Window

## Saving a plot as a PDF file

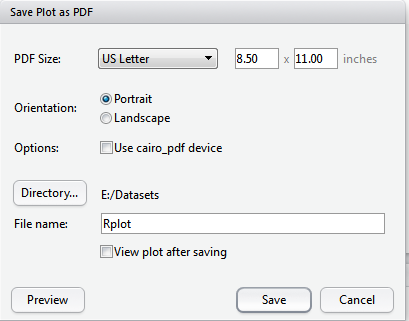
Click on and arrow to expand the Export menu on Figure 57. Select Save as PDF.



Select Save as PDF

Figure 57: Select Save as PDF

The Save Plot as PDF window on Figure 58 will open. Follow the steps to specify the page size, the file name, and the directory. Click Save to continue.



1. Select the Page Size
2. Select Page Orientation
3. Browse to the folder here you want to save the image (the default path is a working directory.
4. Enter the name or keep the default name
5. Click Save to continue.

Figure 58: Save Plot as PDF Window

## Additional Plot Examples

|  |  |
| --- | --- |
| Pie Chart – shows the frequency or a relative frequency for the discrete values of the categorical variable.  pie(table (cars$Bodystyle)) |  |
| Histogram  hist(cars$Hp, col="blue")  x-axis attribute name  Data frame name  Bars color |  |
| Boxplots  boxplot(cars$Height, col="maroon") |  |
| Step plot  plot(table(cars$Cylinders), type="s", col="dark red") |  |
| Line plot  plot(table(cars$Bodystyle), type="l", col="dark orange") |  |
| Points Plot  plot(table(cars$Bodystyle), type="p", col="dark magenta") |  |
| Lines and Points Plot  plot(table(cars$Bodystyle), type="b", col="cyan") |  |