**Background**

My colleague wanted to be able to do some simple analysis around health  
care using the Centers for Disease Control and Prevention  
([https://www.cdc.gov](https://www.cdc.gov/)), National Center for Health Statistics  
(<https://www.cdc.gov/nchs/index.htm>), National Health Interview Survey  
(<https://www.cdc.gov/nchs/nhis/nhis_2016_data_release.htm>). They  
wanted a series of cross tabulated sets of summary data for variable  
pairings (for example whether or not the respondent had a formal health  
care provider by region of the country). They wanted one Excel  
“workbook” with 12 worksheets each one of which was the summary of  
counts for a pair of variables. From there they could use Excel’s native  
plotting tools to make the graphs they needed.

A little sleuthing around CRAN helped me discover openxlsx which seems  
to be quite active, well maintained, and have a variety of features I  
would need. In my case that involved a download and install first (but  
I’ll comment it out in this version for you). As long as I’m at it  
I’ll load dplyr and ggplot2 (they’ll figure more prominently in my  
next post).

knitr::opts\_chunk$set(echo = TRUE, warning = FALSE)

library(dplyr)

##

## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':

##

## filter, lag

## The following objects are masked from 'package:base':

##

## intersect, setdiff, setequal, union

library(ggplot2)

theme\_set(theme\_bw()) # set theme to my personal preference

# install.packages("openxlsx")

require(openxlsx)

## Loading required package: openxlsx

**Download and structure the data**

The data we wanted was from 2014 and to  
provide the URL for a compressed zipfile  
<ftp://ftp.cdc.gov/pub/Health_Statistics/NCHS/Datasets/NHIS/2016/personsxcsv.zip>.  
Full documentation about the file and its contents and the methodology  
is here <https://www.cdc.gov/nchs/nhis/nhis_2016_data_release.htm>.

After downloading and uncompressing it into a local project directory  
(it’s about 77Mb of data, with over 100,000 rows and more than 600  
columns). I could go to work on processing it to get what was really  
needed for my colleague.

FullFile <- read.csv(file = "personsx.csv")

dim(FullFile)

## [1] 103789 606

# 606 variables too many whittle it down with wild cards

**A little dplyr to make our life easier**

While strictly speaking nothing in the next few steps requires dplyr  
(they can all be done in base R) I will showcase a couple ways dplyr  
can make your data analysis faster and easier.

First off, call me old-fashioned but, I abhor having lots of data in  
working memory that I know for a fact I’ll never use. While my Mac has  
plenty of space let’s showcase dplyr’s ability to help us rapidly  
reduce down to a more manageable dataset. A quick look at the data  
dictionary provided by the **CDC** shows we’ll never use any of the  
variables in the dataset that start with “L” or with “INT” so let’s make  
them go away.

FullFile <- select(FullFile, -starts\_with("L"))

FullFile <- select(FullFile, -starts\_with("INT"))

dim(FullFile)

## [1] 103789 289

# 289 is still big but ...

The other thing we notice is that the variables are all coded as integers when  
we know they are truly factors. Since we know we’re going to analyze  
them as factors let’s take a moment to recode which will also mean an  
opportunity to provide more user friendly labels so we’re not constantly  
referring to the code book to see what “1” really represents in the  
data.

The function is recode\_factor and allows us to map factor labels onto  
the integers to provide a more useful dataset.

FullFile$REGION <- recode\_factor(FullFile$REGION,

`1` = "Northeast",

`2` = "Midwest",

`3` = "South",

`4` = "West")

FullFile$SEX <- recode\_factor(FullFile$SEX, `1` = "Male", `2` = "Female")

FullFile$RACERPI2 <- recode\_factor(FullFile$RACERPI2,

`1` = "White only",

`2` = "Black/African American only",

`3` = "AIAN only",

`4` = "Asian only",

`5` = "Race group not releasable",

`6` = "Multiple race")

FullFile$PDMED12M <- recode\_factor(FullFile$PDMED12M, `1` = "Yes", `2` = "No")

## Warning: Unreplaced values treated as NA as .x is not compatible. Please

## specify replacements exhaustively or supply .default

summary(FullFile$PDMED12M)

## Yes No NA's

## 6744 96986 59

Uh oh, what are we being warned about? recode\_factor has a nice  
rational solution for cases where we don’t specify all the possible  
choices for a variable (see ?recode\_factor) it simply assigns them an  
NA value. In our case that’s just what we want.

Finishing the rest (and suppressing the  
warnings)…

FullFile$PNMED12 <- recode\_factor(FullFile$PNMED12, `1` = "Yes", `2` = "No")

FullFile$PNMED12M <- recode\_factor(FullFile$PNMED12M, `1` = "Yes", `2` = "No")

FullFile$NOTCOV <- recode\_factor(FullFile$NOTCOV, `1` = "Not covered", `2` = "Covered")

FullFile$COVER <- recode\_factor(FullFile$COVER,

`1` = "Private",

`2` = "Medicaid and other public",

`3` = "Other coverage",

`4` = "Uninsured",

`5` = "Do not know")

FullFile$PLNWRKS1 <- recode\_factor(FullFile$PLNWRKS1,

`1` = "Through employer",

`2` = "Through union",

`3` = "Through workplace, but don't know if employer or union",

`4` = "Through workplace, self-employed or professional association",

`5` = "Purchased directly",

`6` = "Through Healthcare.gov or the Affordable Care Act",

`7` = "Through a state/local government or community program",

`8` = "Other",

`9` = "Through school",

`10` = "Through parents",

`11` = "Through relative other than parents")

FullFile$HCSPFYR <- recode\_factor(FullFile$HCSPFYR,

`0` = "Zero",

`1` = "Less than $500",

`2` = "$500 - $1,999",

`3` = "$2,000 - $2,999",

`4` = "$3,000 - $4,999",

`5` = "$5,000 or more")

FullFile$MEDBILL <- recode\_factor(FullFile$MEDBILL, `1` = "Yes", `2` = "No")

FullFile$MEDBPAY <- recode\_factor(FullFile$MEDBPAY, `1` = "Yes", `2` = "No")

# I am thinking that earnings can be collapsed into three attributes: low; medium; high

FullFile$EARNINGS <- recode\_factor(FullFile$ERNYR,

`1` = "$01-$34,999",

`2` = "$01-$34,999",

`3` = "$01-$34,999",

`4` = "$01-$34,999",

`5` = "$01-$34,999",

`6` = "$01-$34,999",

`7` = "$35,000-$74,999",

`8` = "$35,000-$74,999",

`9` = "$35,000-$74,999",

`10` = "$35,000-$74,999",

`11` = "$75,000 and over")

# Education the same: low; medium; high

FullFile$EDUCATION <- recode\_factor(FullFile$EDUC1,

`0` = "HSchool Grad or less",

`1` = "HSchool Grad or less",

`2` = "HSchool Grad or less",

`3` = "HSchool Grad or less",

`4` = "HSchool Grad or less",

`5` = "HSchool Grad or less",

`6` = "HSchool Grad or less",

`7` = "HSchool Grad or less",

`8` = "HSchool Grad or less",

`9` = "HSchool Grad or less",

`10` = "HSchool Grad or less",

`11` = "HSchool Grad or less",

`12` = "HSchool Grad or less",

`13` = "HSchool Grad or less",

`14` = "HSchool Grad or less",

`15` = "Some college or AA degree",

`16` = "Some college or AA degree",

`17` = "Some college or AA degree",

`18` = "Bachelor's or higher",

`19` = "Bachelor's or higher",

`20` = "Bachelor's or higher",

`21` = "Bachelor's or higher")

# Age also collapsed: low; medium; high

FullFile$AGE <- cut(FullFile$AGE\_P,

breaks = c(-Inf, 18, 61, Inf),

labels = c("Less than 18", "18 to 60", "More than 60"),

right = FALSE)

table(FullFile$EDUC1,FullFile$EDUCATION)

##

## HSchool Grad or less Some college or AA degree Bachelor's or higher

## 0 3424 0 0

## 1 1679 0 0

## 2 1633 0 0

## 3 1787 0 0

## 4 1716 0 0

## 5 1822 0 0

## 6 2726 0 0

## 7 1860 0 0

## 8 2644 0 0

## 9 2908 0 0

## 10 2884 0 0

## 11 3186 0 0

## 12 1774 0 0

## 13 2290 0 0

## 14 18413 0 0

## 15 0 14822 0

## 16 0 5413 0

## 17 0 2966 0

## 18 0 0 13883

## 19 0 0 5956

## 20 0 0 955

## 21 0 0 1018

## 96 0 0 0

## 97 0 0 0

## 98 0 0 0

## 99 0 0 0

One easy way to see whether your recoding has had the desired effect is  
to make a simple table that maps the original values to the new values.  
That’s what I’ve done here for **EDUCATION** which shows the collapse of  
categories and the fact that values like 99 have been mapped to NA.

Finally let’s grab just the variables we really need including the newly  
recoded versions and make them a new dataset and take the full version  
out of  
memory.

OfInterest <- select(FullFile, AGE, REGION, SEX, EDUCATION, EARNINGS, PDMED12M, PNMED12M, NOTCOV, MEDBILL)

str(OfInterest)

## 'data.frame': 103789 obs. of 9 variables:

## $ AGE : Factor w/ 3 levels "Less than 18",..: 2 2 2 1 1 2 2 2 3 2 ...

## $ REGION : Factor w/ 4 levels "Northeast","Midwest",..: 3 4 4 4 4 4 3 3 4 3 ...

## $ SEX : Factor w/ 2 levels "Male","Female": 1 2 1 1 2 2 2 1 1 1 ...

## $ EDUCATION: Factor w/ 3 levels "HSchool Grad or less",..: 2 1 3 1 NA 3 1 1 3 2 ...

## $ EARNINGS : Factor w/ 3 levels "$01-$34,999",..: 1 2 2 NA NA 3 1 NA 2 3 ...

## $ PDMED12M : Factor w/ 2 levels "Yes","No": 2 1 2 2 2 2 2 2 2 2 ...

## $ PNMED12M : Factor w/ 2 levels "Yes","No": 2 2 2 2 2 2 2 2 2 2 ...

## $ NOTCOV : Factor w/ 2 levels "Not covered",..: 2 2 2 2 2 2 2 2 2 2 ...

## $ MEDBILL : Factor w/ 2 levels "Yes","No": 2 1 1 1 1 2 2 2 2 2 ...

summary(OfInterest)

## AGE REGION SEX

## Less than 18:25680 Northeast:16883 Male :50257

## 18 to 60 :58025 Midwest :20507 Female:53532

## More than 60:20084 South :35826

## West :30573

## EDUCATION EARNINGS PDMED12M

## HSchool Grad or less :50746 $01-$34,999 :20390 Yes : 6744

## Some college or AA degree:23201 $35,000-$74,999 :12927 No :96986

## Bachelor's or higher :21812 $75,000 and over: 6413 NA's: 59

## NA's : 8030 NA's :64059

## PNMED12M NOTCOV MEDBILL

## Yes : 4978 Not covered:10506 Yes :16282

## No :98741 Covered :92181 No :87096

## NA's: 70 NA's : 1102 NA's: 411

##

rm(FullFile)

**Driving Excel**

Everything we have done so far has been in preparation for actually  
providing my colleague the data she wanted in Excel so now on to the  
main event. In this post I am deliberately going to **NOT** build  
functions or use ggplot to make the graphs she wanted. Those will be  
topics for future posts.

You may have noticed that a table is almost exactly what she wants. If  
the variables of interest are AGE and whether or not the person has  
health care coverage (NOTCOV) then the command would be  
table(OfInterest$AGE,OfInterest$NOTCOV) and we have results we need.  
The openxlsx package provides a write.xlsx function that will accept the  
table command and produce a properly formatted workbook as output. The  
code below will produce SimpleExcelExample.xlsx in your working  
directory. The number of options is legion and ?write.xlsx will  
display them for you.

table(OfInterest$AGE,OfInterest$NOTCOV)

##

## Not covered Covered

## Less than 18 1312 24193

## 18 to 60 8730 48484

## More than 60 464 19504

write.xlsx(table(OfInterest$AGE,OfInterest$NOTCOV), file = "SimpleExcelExample.xlsx")

**Finishing up the manual way**

The last thing I will describe in this post is how to make one workbook  
with multiple sheets. Where each sheet represents one variable pairing  
(e.g. AGE by NOTCOV). We’ll even label the tabs (worksheets) in a  
thoughtful way. So my colleague can easily see which sheet corresponds  
to which variable pairing.

The code below follows this pattern

1. Create a new empty workbook object wb <- createWorkbook()
2. Invent a name for the **tab** or worksheet inside the workbook  
   NameofSheet
3. Make a table for a pair of variables TheData <-  
   table(OfInterest$EDUCATION,OfInterest$NOTCOV)
4. Add a worksheet (tab) into the workbook addWorksheet
5. Write the table we made onto the worksheet with writeData
6. Repeat steps 2 through 5 **12** times

### Manual and painful way

## Create a new workbook

wb <- createWorkbook()

# education by each of the other 4 variables

NameofSheet <- "CoverageByEducation"

TheData <- table(OfInterest$EDUCATION,OfInterest$NOTCOV)

addWorksheet(wb = wb, sheetName = NameofSheet)

writeData(wb = wb, sheet = NameofSheet, x = TheData, borders = "n")

NameofSheet <- "MedbillByEducation"

TheData <- table(OfInterest$EDUCATION,OfInterest$MEDBILL)

addWorksheet(wb = wb, sheetName = NameofSheet)

writeData(wb = wb, sheet = NameofSheet, x = TheData, borders = "n")

NameofSheet <- "PDMED12MByEducation"

TheData <- table(OfInterest$EDUCATION,OfInterest$PDMED12M)

addWorksheet(wb = wb, sheetName = NameofSheet)

writeData(wb = wb, sheet = NameofSheet, x = TheData, borders = "n")

NameofSheet <- "PNMED12MByEducation"

TheData <- table(OfInterest$EDUCATION,OfInterest$PNMED12M)

addWorksheet(wb = wb, sheetName = NameofSheet)

writeData(wb = wb, sheet = NameofSheet, x = TheData, borders = "n")

# earnings by each of the other 4 variables

NameofSheet <- "CoverageByEarnings"

TheData <- table(OfInterest$EARNINGS,OfInterest$NOTCOV)

addWorksheet(wb = wb, sheetName = NameofSheet)

writeData(wb = wb, sheet = NameofSheet, x = TheData, borders = "n")

NameofSheet <- "MedbillByEarnings"

TheData <- table(OfInterest$EARNINGS,OfInterest$MEDBILL)

addWorksheet(wb = wb, sheetName = NameofSheet)

writeData(wb = wb, sheet = NameofSheet, x = TheData, borders = "n")

NameofSheet <- "PDMED12MByEarnings"

TheData <- table(OfInterest$EARNINGS,OfInterest$PDMED12M)

addWorksheet(wb = wb, sheetName = NameofSheet)

writeData(wb = wb, sheet = NameofSheet, x = TheData, borders = "n")

NameofSheet <- "PNMED12MByEarnings"

TheData <- table(OfInterest$EARNINGS,OfInterest$PNMED12M)

addWorksheet(wb = wb, sheetName = NameofSheet)

writeData(wb = wb, sheet = NameofSheet, x = TheData, borders = "n")

# age by each of the other 4 variables

NameofSheet <- "CoverageByAge"

TheData <- table(OfInterest$AGE,OfInterest$NOTCOV)

addWorksheet(wb = wb, sheetName = NameofSheet)

writeData(wb = wb, sheet = NameofSheet, x = TheData, borders = "n")

NameofSheet <- "MedbillByAge"

TheData <- table(OfInterest$AGE,OfInterest$MEDBILL)

addWorksheet(wb = wb, sheetName = NameofSheet)

writeData(wb = wb, sheet = NameofSheet, x = TheData, borders = "n")

NameofSheet <- "PDMED12MByAge"

TheData <- table(OfInterest$AGE,OfInterest$PDMED12M)

addWorksheet(wb = wb, sheetName = NameofSheet)

writeData(wb = wb, sheet = NameofSheet, x = TheData, borders = "n")

NameofSheet <- "PNMED12MByAge"

TheData <- table(OfInterest$AGE,OfInterest$PNMED12M)

addWorksheet(wb = wb, sheetName = NameofSheet)

writeData(wb = wb, sheet = NameofSheet, x = TheData, borders = "n")

#

saveWorkbook(wb, "BetterExcelExample.xlsx", overwrite = TRUE) ## save to working directory

**All done (not really)**

Hopefully this post helps you understand how to use the openxlsx  
package to have R drive Excel to help you with your data analysis. In  
my next post I’ll build on this scaffolding to discuss how to make these  
very same graphs in ggplot2 (which IMHO runs circles around Excel for  
scientific plotting), as well as making this all more efficient through  
the use of functions to take care of some of the more repetitive chores.

**xlsx package** is one of the powerful R packages to **read**, **write** and **format** **Excel files**. It is a java-based solution and it is available for Windows, Mac and Linux. It works for both **Excel 2007** and **Excel 97/2000/XP/2003** file formats (**xls** and **xlsx** file formats).

Unfortunately the xlsx package is not especially well documented.

This article is a quick start guide to manipulate **Excel files** in **R** using **xlsx package** .

In this tutorial, you will learn how to use **xlsx package** to :

* **Read** and **write** Excel files
* **Add data sets** and **images** (or plots) into an Excel worksheet
* **Format the appearance** of the Excel worksheet by setting data formats, fonts, colors and borders

**Install and load xlsx package**

install.packages("xlsx")

library("xlsx")

Note that, xlsx packages depends on **rJava** and **xlsxjars** R packages.

**Read an Excel file**

The R functions **read.xlsx()** and **read.xlsx2()** can be used to **read** the contents of an **Excel** worksheet into an **R** data.frame.

The difference between these two functions is that :

* read.xlsx preserves the data type. It tries to guess the class type of the variable corresponding to each column in the worksheet. Note that, **read.xlsx** function is slow for large data sets (worksheet with more than 100 000 cells).
* read.xlsx2 is faster on big files compared to read.xlsx function.

The simplified formats of these two functions are:

read.xlsx(file, sheetIndex, header=TRUE, colClasses=NA)

read.xlsx2(file, sheetIndex, header=TRUE, colClasses="character")

* **file** : the path to the file to read
* **sheetIndex** : a number indicating the index of the sheet to read; e.g : use sheetIndex=1 to read the first sheet
* **header** : a logical value. If TRUE, the first row is used as the names of the variables
* **colClasses** : a character vector that represents the class of each column

Examples :

library(xlsx)

file <- system.file("tests", "test\_import.xlsx", package = "xlsx")

res <- read.xlsx(file, 1) # read first sheet

head(res[, 1:6])

NA. Population Income Illiteracy Life.Exp Murder

1 Alabama 3615 3624 2.1 69.05 15.1

2 Alaska 365 6315 1.5 69.31 11.3

3 Arizona 2212 4530 1.8 70.55 7.8

4 Arkansas 2110 3378 1.9 70.66 10.1

5 California 21198 5114 1.1 71.71 10.3

6 Colorado 2541 4884 0.7 72.06 6.8

Note that read.xlsx and read.xlsx2 functions can be used to read both .xls and .xlsx file formats.

**Write data to an Excel file**

The R functions **write.xlsx()** and **write.xlsx2()** can be used to **export** data from **R** to an **Excel** workbook. Note that **write.xlsx2** achieves better performance compared to **write.xlsx** for very large data.frame (with more than 100 000 cells).

The simplified formats of these two functions are:

write.xlsx(x, file, sheetName="Sheet1",

col.names=TRUE, row.names=TRUE, append=FALSE)

write.xlsx2(x, file, sheetName="Sheet1",

col.names=TRUE, row.names=TRUE, append=FALSE)

* **x** : a data.frame to be written into the workbook
* **file** : the path to the output file
* **sheetName** : a character string to use for the sheet name.
* **col.names, row.names** : a logical value specifying whether the column names/row names of x are to be written to the file
* **append** : a logical value indicating if x should be appended to an existing file.

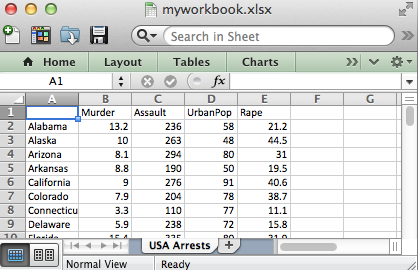
Examples :

library(xlsx)

write.xlsx(USArrests, file="myworkbook.xlsx",

sheetName="USA Arrests")

Note that, the above code saves the Excel file in your current working directory.



To **add multiple data sets** in the same Excel workbook, you have to use the argument **append = TRUE**. This is illustrated in the following R code :

# Write the first data set in a new workbook

write.xlsx(USArrests, file="myworkbook.xlsx",

sheetName="USA-ARRESTS", append=FALSE)

# Add a second data set in a new worksheet

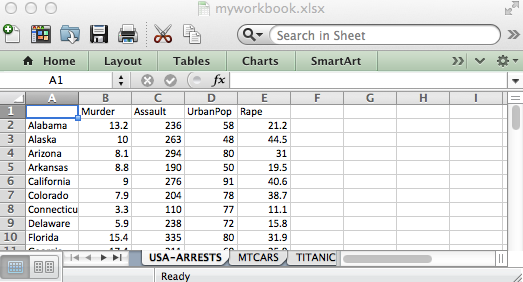
write.xlsx(mtcars, file="myworkbook.xlsx", sheetName="MTCARS",

append=TRUE)

# Add a third data set

write.xlsx(Titanic, file="myworkbook.xlsx", sheetName="TITANIC",

append=TRUE)



As you can see from the image above, it’s possible to add multiple data sets in the same Excel file. However, the method is very repetitive. You will find in the next section a simple function to add different types of data in a single call.

**Simple R function to export quickly multiple data sets to the same Excel workbook**

This section provides an R function to easily **export multiple R objects to an Excel Workbook in a single call**. The different objects (data) are written in different worksheets from the same Excel workbook. The object names are used for naming the different sheets.

The R code of the function is :

#+++++++++++++++++++++++++++

# xlsx.writeMultipleData

#+++++++++++++++++++++++++++++

# file : the path to the output file

# ... : a list of data to write to the workbook

xlsx.writeMultipleData <- function (file, ...)

{

require(xlsx, quietly = TRUE)

objects <- list(...)

fargs <- as.list(match.call(expand.dots = TRUE))

objnames <- as.character(fargs)[-c(1, 2)]

nobjects <- length(objects)

for (i in 1:nobjects) {

if (i == 1)

write.xlsx(objects[[i]], file, sheetName = objnames[i])

else write.xlsx(objects[[i]], file, sheetName = objnames[i],

append = TRUE)

}

}

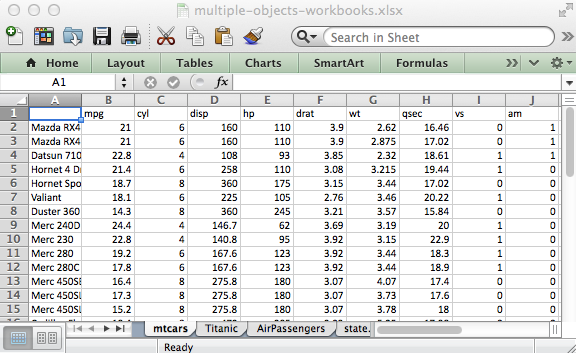
The function **xlsx.writeMultipleData** works for data frames, matrices, time series, and tables.

**Example of usage :**

Use the R code below to save mtcars (a data frame), Titanic (a table), AirPassengers (a time series) and state.x77 (a matrix) :

xlsx.writeMultipleData("myworkbook.xlsx",

mtcars, Titanic, AirPassengers, state.x77)



**Create and format a nice Excel workbook**

The function **write.xlsx()** is useful when you want just to write a data.frame to an xlsx file. The goal of this section is to show you how to create a nice Excel report containing a formatted data table and plots.

The following steps are required :

1. Create a workbook
2. Define some **cell styles** : **Font** color and size, text **alignment**, **border** and **data format**, …
3. Write a table into an Excel spreadsheet using the defined styles in step 2.
4. Save the workbook to a file
5. Open and view the resulting workbook

In the next sections, I will show you step by step how to change the appearance of Excel worksheet in R. Note that, formatting Excel worksheets using **xlsx R package** requires some hard coding. This is why, I recently implemented the **r2excel package** which depends on **xlsx** package and it provides an **easy to use functions** to quickly import data from Excel files and to create a nice

**Step 1/5. Create a new Excel workbook**

The function **createWorkbook()** can be used. It works for both **.xls** and **.xlsx** file formats.

# create a new workbook for outputs

# possible values for type are : "xls" and "xlsx"

wb<-createWorkbook(type="xlsx")

**Step 2/5. Define some cell styles for formating the workbook**

We’ll define some cell styles to change :

* the appearance of the **sheet title**
* the appearance of the **row and column names** of the data table
* the **text alignment** for the table column names
* the cell borders around the column names

The R function **CellStyle()** can be used to create cell styles. A simplified format of the function is :

CellStyle(wb, dataFormat=NULL, alignment=NULL,

border=NULL, fill=NULL, font=NULL)

<-="" cellstyle(wb)="" +="" font(wb,="" heightinpoints="14," isitalic="TRUE," #="" styles="" for="" the="" data="" table="" row="" column="" names="" table\_rownames\_style="" table\_colnames\_style="" alignment(wraptext="TRUE," horizontal="ALIGN\_CENTER" )="" border(color="black" ,="" position="c(" top","="" "bottom"),="" pen="c(" border\_thin","="" "border\_thick"))="" ```=""

* **wb** : a workbook object as returned by createWorkbook or loadWorkbook.
* **dataFormat** : a *DataFormat* object
* **alignment** : a *Alignment* object
* **border** : a *Border* object
* **font** : a *Font* object

# Define some cell styles

#++++++++++++++++++++

# Title and sub title styles

TITLE\_STYLE <- CellStyle(wb)+ Font(wb, heightInPoints=16,

color="blue", isBold=TRUE, underline=1)

SUB\_TITLE\_STYLE <- CellStyle(wb) +

Font(wb, heightInPoints=14,

isItalic=TRUE, isBold=FALSE)

# Styles for the data table row/column names

TABLE\_ROWNAMES\_STYLE <- CellStyle(wb) + Font(wb, isBold=TRUE)

TABLE\_COLNAMES\_STYLE <- CellStyle(wb) + Font(wb, isBold=TRUE) +

Alignment(wrapText=TRUE, horizontal="ALIGN\_CENTER") +

Border(color="black", position=c("TOP", "BOTTOM"),

pen=c("BORDER\_THIN", "BORDER\_THICK"))

1. **wb** : a workbook object as returned by createWorkbook or loadWorkbook.
2. **The main arguments for Font() function** :
   * **color** : font color
   * **heightInPoints** : *font size*. Usual values are 10, 12, 14, etc
   * **isBold, isItalic** : a logical indicating whether the font should be *bold* or *italic*
   * **underline** : an integer specifying the *thickness of the underline*. Possible values are 0, 1, 2.
   * **name** : the font to use; e.g: “Courier New”.
3. **The main arguments for Alignment() function** :
   * **wrapText** : a logical indicating whether the text should be wrapped.
   * **horizontal** : the *horizontal alignment*. Possible values are : “ALIGN\_CENTER”, “ALIGN\_JUSTIFY”, “ALIGN\_LEFT”, “ALIGN\_RIGHT”.
   * **vertical** : the *vertical alignment*. Possible values are : “VERTICAL\_BOTTOM”, “VERTICAL\_CENTER”, “VERTICAL\_JUSTIFY”, “VERTICAL\_TOP”
   * **rotation** : a numerical value specifying the degrees you want to *rotate the text* in the cell. Default value is 0.
4. **The main arguments for Border() function** :
   * **color** : the *border color*; e.g : color=“red” or color =“#FF0000”
   * *position* : the *border position*. Allowed values are : “BOTTOM”, “LEFT”, “TOP”, “RIGHT”
   * **pen** : the pen style. Allowed values are : “BORDER\_DASH\_DOT”, “BORDER\_DASH\_DOT\_DOT”, “BORDER\_DASHED”, “BORDER\_DOTTED”, “BORDER\_DOUBLE”, “BORDER\_HAIR”, “BORDER\_MEDIUM”, “BORDER\_MEDIUM\_DASH\_DOT”, “BORDER\_MEDIUM\_DASH\_DOT\_DOT”, “BORDER\_MEDIUM\_DASHED”, “BORDER\_NONE”, “BORDER\_SLANTED\_DASH\_DOT”, “BORDER\_THICK”, “BORDER\_THIN”.

**Step 3/5. Write data and plots into the workbook**

**Create a new sheet in the workbook**

To add data, the first step is to create a sheet in the workbook to contain the data. This can be done using the function **creatSheet()** :

# Create a new sheet in the workbook

sheet <- createSheet(wb, sheetName = "US State Facts")

**Add a title into a worksheet**

To add a title, the procedure is :

1. create a new row
2. create a cell in this row to contain the title.
3. set the cell value.

To simplify the R code, I wrote a helper function for adding a title :

#++++++++++++++++++++++++

# Helper function to add titles

#++++++++++++++++++++++++

# - sheet : sheet object to contain the title

# - rowIndex : numeric value indicating the row to

#contain the title

# - title : the text to use as title

# - titleStyle : style object to use for title

xlsx.addTitle<-function(sheet, rowIndex, title, titleStyle){

rows <-createRow(sheet,rowIndex=rowIndex)

sheetTitle <-createCell(rows, colIndex=1)

setCellValue(sheetTitle[[1,1]], title)

setCellStyle(sheetTitle[[1,1]], titleStyle)

}

Copy and paste the code of the function xlsx.addTitle into your R console before continuing.

# Add title

xlsx.addTitle(sheet, rowIndex=1, title="US State Facts",

titleStyle = TITLE\_STYLE)

# Add sub title

xlsx.addTitle(sheet, rowIndex=2,

title="Data sets related to the 50 states of USA.",

titleStyle = SUB\_TITLE\_STYLE)

**Add a table into a worksheet**

The function **addDataframe()** can be used to add the table in the new sheet.

state.x77 data table is used in the following example :

head(state.x77)

Population Income Illiteracy Life Exp Murder HS Grad Frost Area

Alabama 3615 3624 2.1 69.05 15.1 41.3 20 50708

Alaska 365 6315 1.5 69.31 11.3 66.7 152 566432

Arizona 2212 4530 1.8 70.55 7.8 58.1 15 113417

Arkansas 2110 3378 1.9 70.66 10.1 39.9 65 51945

California 21198 5114 1.1 71.71 10.3 62.6 20 156361

Colorado 2541 4884 0.7 72.06 6.8 63.9 166 103766

# Add a table

addDataFrame(state.x77, sheet, startRow=3, startColumn=1,

colnamesStyle = TABLE\_COLNAMES\_STYLE,

rownamesStyle = TABLE\_ROWNAMES\_STYLE)

# Change column width

setColumnWidth(sheet, colIndex=c(1:ncol(state.x77)), colWidth=11)

* **Arguments for addDataFrame() function** :
  + **startRow**, **startColumn** : a numeric value indicating the starting row and column
  + **colnameStyle**, **rownameStyle** : A CellStyle object to customize the table header and row names
* **Arguments for setColumnWidth() function** :
  + **colIndex** : a numeric vector indicating the columns you want to change the size.
  + **colWidth** : the width of the column

**Add a plot into an Excel worksheet**

# create a png plot

png("boxplot.png", height=800, width=800, res=250, pointsize=8)

boxplot(count ~ spray, data = InsectSprays,

col = "blue")

dev.off()

# Create a new sheet to contain the plot

sheet <-createSheet(wb, sheetName = "boxplot")

# Add a title to the sheet

xlsx.addTitle(sheet, rowIndex=1,

title="Box plot using InsectSprays data",

titleStyle = TITLE\_STYLE)

# Add the plot created previously

addPicture("boxplot.png", sheet, scale = 1, startRow = 4,

startColumn = 1)

# Remove the plot from the disk

res<-file.remove("boxplot.png")

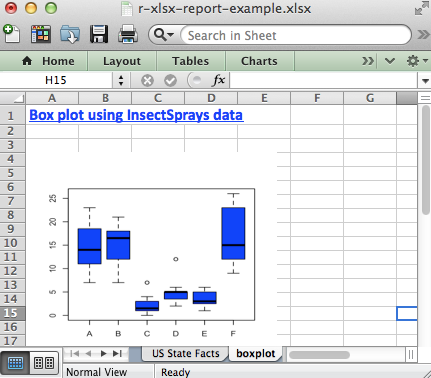
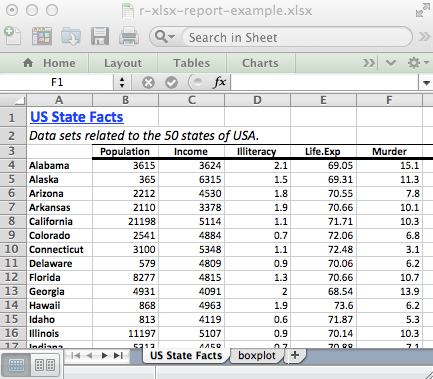
**Step 4/5. Save the Excel workbook to the disk**

# Save the workbook to a file

saveWorkbook(wb, "r-xlsx-report-example.xlsx")

**Step 5/5. Open and view the resulting Excel workbook**

Go to your current working directory and open the created workbook.



**The complete R script to create a nice Excel report**

The complete R script to create the workbook above is :

library(xlsx)

# create a new workbook for outputs

#++++++++++++++++++++++++++++++++++++

# possible values for type are : "xls" and "xlsx"

wb<-createWorkbook(type="xlsx")

# Define some cell styles

#++++++++++++++++++++++++++++++++++++

# Title and sub title styles

TITLE\_STYLE <- CellStyle(wb)+ Font(wb, heightInPoints=16,

color="blue", isBold=TRUE, underline=1)

SUB\_TITLE\_STYLE <- CellStyle(wb) +

Font(wb, heightInPoints=14,

isItalic=TRUE, isBold=FALSE)

# Styles for the data table row/column names

TABLE\_ROWNAMES\_STYLE <- CellStyle(wb) + Font(wb, isBold=TRUE)

TABLE\_COLNAMES\_STYLE <- CellStyle(wb) + Font(wb, isBold=TRUE) +

Alignment(wrapText=TRUE, horizontal="ALIGN\_CENTER") +

Border(color="black", position=c("TOP", "BOTTOM"),

pen=c("BORDER\_THIN", "BORDER\_THICK"))

# Create a new sheet in the workbook

#++++++++++++++++++++++++++++++++++++

sheet <- createSheet(wb, sheetName = "US State Facts")

#++++++++++++++++++++++++

# Helper function to add titles

#++++++++++++++++++++++++

# - sheet : sheet object to contain the title

# - rowIndex : numeric value indicating the row to

#contain the title

# - title : the text to use as title

# - titleStyle : style object to use for title

xlsx.addTitle<-function(sheet, rowIndex, title, titleStyle){

rows <-createRow(sheet,rowIndex=rowIndex)

sheetTitle <-createCell(rows, colIndex=1)

setCellValue(sheetTitle[[1,1]], title)

setCellStyle(sheetTitle[[1,1]], titleStyle)

}

# Add title and sub title into a worksheet

#++++++++++++++++++++++++++++++++++++

# Add title

xlsx.addTitle(sheet, rowIndex=1, title="US State Facts",

titleStyle = TITLE\_STYLE)

# Add sub title

xlsx.addTitle(sheet, rowIndex=2,

title="Data sets related to the 50 states of USA.",

titleStyle = SUB\_TITLE\_STYLE)

# Add a table into a worksheet

#++++++++++++++++++++++++++++++++++++

addDataFrame(state.x77, sheet, startRow=3, startColumn=1,

colnamesStyle = TABLE\_COLNAMES\_STYLE,

rownamesStyle = TABLE\_ROWNAMES\_STYLE)

# Change column width

setColumnWidth(sheet, colIndex=c(1:ncol(state.x77)), colWidth=11)

# Add a plot into a worksheet

#++++++++++++++++++++++++++++++++++++

# create a png plot

png("boxplot.png", height=800, width=800, res=250, pointsize=8)

boxplot(count ~ spray, data = InsectSprays,

col = "blue")

dev.off()

# Create a new sheet to contain the plot

sheet <-createSheet(wb, sheetName = "boxplot")

# Add title

xlsx.addTitle(sheet, rowIndex=1, title="Box plot using InsectSprays data",

titleStyle = TITLE\_STYLE)

# Add the plot created previously

addPicture("boxplot.png", sheet, scale = 1, startRow = 4,

startColumn = 1)

# remove the plot from the disk

res<-file.remove("boxplot.png")

# Save the workbook to a file...

#++++++++++++++++++++++++++++++++++++

saveWorkbook(wb, "r-xlsx-report-example.xlsx")

**Infos**

This analysis has been performed using R (ver. 3.1.0).