PhUSE 2014

Paper AD04

R for SAS programmers: It's different, but friendly

Friedrich Schuster, HMS Analytical Software GmbH, Heidelberg, Germany

ABSTRACT

In recent years, a large number of pharmaceutical companies have adopted R as a data analysis tool. The main reasons for the increasing importance of R are the availability and ever-growing number of high-quality statistical methods, very good graphics capabilities and a wide range of useful programming extensions. This presentation addresses SAS™ users who are new to R and want to learn more about R. It gives a brief overview of the characteristics of R as a programming language and as an environment for statistical computing. Some particularly important differences between R and SAS will be highlighted. Small code examples illustrate solutions for common SAS data management and analysis tasks. It also points to further sources of information, such that participants will be able to continue learning R on their own by the end of this presentation.

INTRODUCTION

In recent years, more and more pharmaceutical companies have adopted R for data management and analysis. Currently R and SAS™ are often used side by side for similar tasks, and because of this, R becomes also more important for SAS programmers when it comes to understand, use and maintain code of their colleagues.

This paper addresses SAS users who are new to R and want to learn more about R. It gives a brief overview of the history and main characteristics of R. Small code examples from a (fictional) blood pressure study scenario illustrate solutions for common data management and analysis tasks.

KEY CHARACTERISTICS OF R

HISTORY

In the 70's the "S statistical programming language" was designed at the Bell Laboratories. It was developed primarily by John Chambers with the intent "to turn ideas into software, quickly and faithfully". In 1988, after many changes, the version 3 of the S language ("New S") was released. R is a re-implementation of the original S language and its interpreter. Today R code is still very similar to the "New S" code of 1988, and R has become the reference implementation of the S language.

The R Project itself started in 1993 with an announcement in the "s-news" mailing list, and a group of developers formed up. In 1997 the first alpha version of the "R environment" was released under the Free Software Foundation general license (GNU Public License).

Today R has approximately 3 million users, and more than 5000 extension packages for R have been released.

R AS A PROGRAMMING LANGUAGE

The R programming language has gained a lot of popularity recently. It is currently listed at position 9 of the IEEE Spectrum's Top Programming Language Ranking.

The main features of R are:

- Like SAS it is an interpreted language.
- R has been specifically designed as a language for statistical and mathematical problems, so it is a good example of a domain specific language.
- Like MATLAB or SAS IML, R is a vector language. It has built-in capabilities for the manipulation of vectors and matrices, instead of using loops for data processing.
- In R everything is an object. Common object types are predefined, e.g. data objects, statistical functions, result objects etc. It is also possible to define new objects with object oriented programming techniques.
- R is also a functional language. Functions are objects.
- Most important: R is a highly extensible language.

R AS AN ENVIRONMENT FOR STATISTICAL COMPUTING

Just like SAS code is executed within the SAS System, R code runs within the interpreter of the "R environment for statistical computing". It is possible to execute R programs interactively or as a batch process. The user interface

can be as simple as a unix console window, or as complex as a modern graphical Integrated Development Environment (IDE) such as RStudio.

One of the most important differences between SAS and R is data handling: in R, data objects reside in main memory, whereas SAS reads data sequentially from disk, processes data records, and writes processed data back to a file. Therefore, in SAS only part of the data has to reside in main memory.

Just like SAS, the R environment supports basic data import and export, has its own graphics system and a build in help system.

R AND ITS PACKAGE INFRASTRUCTURE

SAS provides additional products and components for installation in the SAS System (e.g. ETS, IML, Database interfaces etc.). In R additional components are called 'packages' and R contains a very elaborate package management system for installing, using and updating these extensions. Actually, the R environment consists of a small core program with the R interpreter and runtime environment and a set of preinstalled packages for all higher programming functions, statistical algorithms and graphics.

The extensibility of R and (as a result) the availability of a very large number of packages is one of the main strengths of R. Packages usually contain R functions, data files, help files, code examples, PDF manuals, and unit tests.

To work with R, by rule of thumb, you will have to know the basics of the R language (which is quite small), but you will do most of the 'hard work' with specialized functions from R's extension packages.

Packages are freely available and downloadable from the Internet from package repositories such as CRAN. Download and installation is very fast and convenient, and can be done from within the R environment with a single command.

Packages are developed by R users and statisticians. Therefore, most of them are useful and up to date, but sometimes it is difficult to assess the quality of packages. The large majority of packages are released under an open source free software license (mostly under the GPL), so they are free.

SIMILARITIES AND DIFFERENCES BETWEEN R AND SAS

The following points summarize the key similarities and differences between R and SAS:

- Both systems have been designed for statistical programming and analytics.
- The R environment contains only one programming language. In the SAS System multiple "sub-languages" are used (SAS/BASE with its data step, Macro language, IML etc.).
- Both systems can do matrix operations, but in R this feature is already integrated within the language itself.
- R is completely object based, and has some object oriented programming capabilities.
- R is a functional language and functions are objects.
- Data handling in R and SAS differs significantly. R works with data in main memory, SAS uses data files on disk.
- SAS with its data step loops over data files record by record. In R loops are avoided, and vectorized functions work with matrices and vectors.
- R is open source, free of charge and the development of R and its packages is completely community driven. SAS is closed source and owned by a private company. An annual license fee is charged.

CODE EXAMPLES AND RESULTS

The above summary shows that SAS and R are quite different from each other. The following code examples try to give an impression on how to work with R. Common data management and analysis tasks are performed and explained, and references to similar SAS procedures are given. The examples are based on the diastolic blood pressure example from the book "Clinical Trial Data Analysis Using R" (2010) by Din Chen, Karl E. Peace.

DATA ACCESS AND IMPORT

The raw blood pressure data is stored in a CSV formatted file. Import of CSV files into a SAS dataset is done via PROC IMPORT. In R the function 'read.csv()' reads the data file and stores the values as a data frame object in memory. A data frame is the most common R object for tabular data. The name of the data frame is 'data1'. The "arrow" is the standard assignment operator of R.

```
data1 <- read.csv("./data/DiastolicBloodPressure_initial.csv")
```

Similar to SAS R can read data from a large number of data formats and database connections, e.g. Microsoft Excel, Databases and even from SAS binary data files (sas7bdat).

IDENTIFYING MISSING VALUES, DATA MODIFICATION

After the data import the first step is to identify the structure of the variables and checking it for missing values and invalid codes. A simple way to do this in SAS is to use PROC CONTENTS or PROC SQL to get a listing of the variables, and to use PROC PRINT to view the data itself.

In R the 'str()' function lists the variables, variable types and the first few values of a data frame.

```
str(data1)
```

This is the output of the 'str()' function:

```
'data.frame':    40 obs. of  9 variables:
 $ Subject      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Treatment    : Factor w/ 3 levels "","A","B": 2 2 1 2 2 2 2 2 2 2 ...
 $ DBP.Baseline: int  114 116 119 115 116 117 118 120 114 115 ...
 $ DBP.Month1   : int  115 113 115 113 112 112 111 115 112 113 ...
 $ DBP.Month2   : int  NA NA 113 112 107 113 100 113 113 108 ...
 $ DBP.Month3   : int  109 NA 104 109 104 104 109 102 109 106 ...
 $ DBP.Month4   : int  105 101 98 101 105 102 99 102 103 97 ...
 $ Age          : int  43 51 48 42 49 47 50 61 43 51 ...
 $ Sex          : Factor w/ 3 levels "","F","M": 2 3 2 1 3 3 2 3 3 3 ...
```

This data set contains nine variables: The subject ID (Subject), the treatment factor with the factor levels "A" (for Treatment) and "B" (for Placebo), the blood pressure baseline "DBP.Baseline", the blood pressure data for the four months of treatment, and finally "Age" (in years) and "Sex" with the factor levels "F" for female and "M" for male.

Most variables contain integer values. "Treatment" and "Sex" are factor variables. Factor variables consist of an alphanumeric label and a numerical code for each factor level. Other variable types exist in R, e.g. Date and Boolean types.

The 'head()' function then lists the first six records of the data frame. By typing the name of the data frame all records are listed.

```
head(data1)
```

```
  Subject Treatment DBP.Baseline DBP.Month1 DBP.Month2 DBP.Month3 DBP.Month4 Age
1       1         A           114        115         NA        109        105  43
2       2         A           116        113         NA         NA        101  51
3       3                     119        115        113        104         98  48
4       4         A           115        113        112        109        101  42
5       5         A           116        112        107        104        105  49
6       6         A           117        112        113        104        102  47
  Sex
1   F
2   M
3   F
4
5   M
6   M
```

This listing clearly shows two blank entries in the factor variables "Treatment" and "Sex" and a few "NA" entries in the blood pressure variables "DBP.xxx". "NA" is the missing value code for numeric data within R (similar to the "." in SAS). The blank values show where alphanumeric columns of the initial data file were empty. Alphanumeric columns are by default read in as factor variables.

Checking for completeness of cases and identifying incomplete data records is simple with the 'complete.cases()' function in R. This function returns a vector of Boolean values, one for each record, indicating which case is complete or contains NA values. The first two records of the data example are incomplete:

```
complete.cases(data1)
```

```
 [1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[37]  TRUE  TRUE  TRUE  TRUE
```

To select incomplete records from the data, the resulting Boolean vector is used in a filtering / subsetting statement in R. By negating the vector, complete cases are filtered out and only the two first records are listed.

The square brackets after the "data1" object contains the row selection statement and, after the comma, the empty column selection statement. In this example, using the row selection statement is similar to a WHERE statement in SAS. Leaving the column selection statement blank means no column selection.

```
data1[ !complete.cases(data1) , ]
```

```
  Subject Treatment DBP.Baseline DBP.Month1 DBP.Month2 DBP.Month3 DBP.Month4 Age
1       1         A          114        115         NA        109        105  43
2       2         A          116        113         NA         NA        101  51
  Sex
1   F
2   M
```

But the 'complete.cases()' function was not able to find the remaining two empty values in the "Treatment" and "Sex" variables.

First, we have to know how to address a single variable from within the R data frame. To do this, we can access individual columns of a data.frame by using the dollar sign and the name of the variable:

```
data1$Treatment
```

Another more flexible way is to use the column selection statement. The following statement selects all rows and only the column "Treatment":

```
data1[ , "Treatment" ]
```

Then the given variable can be checked for the validity of its codes. For this, the %in% operator is used. In combination with the subsetting / filtering statement, the data records with invalid entries in the "Treatment" variable are identified. Valid labels for Treatment are "A" (for Treatment) and "B" (for Placebo). No blank or missing values are allowed. The 'c()' function concatenates the letters and creates an alphanumeric vector.

```
data1[ !data1$Treatment %in% c("A", "B") , ]
```

```
  Subject Treatment DBP.Baseline DBP.Month1 DBP.Month2 DBP.Month3 DBP.Month4 Age
3       3                   119        115         113        104         98  48
  Sex
3   F
```

To check a continuous range of values, it is possible to use the range operator ":". In this example the age of the subjects is expected to be greater or equal to 35 years and less or equal 65 years. As all age values lie within the required range, no data lines are printed by the following command:

```
data1[ !data1$Age %in% 35:65 , ]
```

Now all records with missing values have been identified. Accessing the values of single record is handled in much the same way as shown above. First, the record has to be selected, for example by the subject ID variable "Subject":

```
data1[ data1$Subject == 1 , ]
```

```
  Subject Treatment DBP.Baseline DBP.Month1 DBP.Month2 DBP.Month3 DBP.Month4 Age
1       1         A          114        115         NA        109        105  43
  Sex
1   F
```

Then the variable with the missing value may be addressed using the "$" sign and the variable name. Then a new values is assigned via the assignment operator "<-".

```
data1[ data1$Subject == 1 , ]$DBP.Month2 <- 113
```

The code for replacing all missing values looks like this:

```
# Replace missing values with valid entries:

data1[ data1$Subject == 1 , ]$DBP.Month2 <- 113
data1[ data1$Subject == 2 , ]$DBP.Month2 <- 112
data1[ data1$Subject == 2 , ]$DBP.Month3 <- 103
data1[ data1$Subject == 3 , ]$Treatment <- "A"
data1[ data1$Subject == 4 , ]$Sex <- "F"
```

As you can see, unlike the SAS data step, R does not loop over the records of a data set. All records and variables are directly accessible in memory.

Finally, the correctness of the modifications should be checked again by listing the structure and records of the modified data frame. For this, we first create and then execute a custom R function:

```
# This creates the R function 'info':

info <- function(d) {
  writeLines("First display  the structure of the data frame.")
  str(d)
  writeLines("Then print the first 6 observations to see the variables and values.")
  print(head(d))
}
```

```
# Call the new 'info' function:

info(data1)
```

The SAS counterpart to custom R functions are SAS Macros or (more recently and more similar) the SAS Function Compiler (FCMP), that enables programmers to create their own SAS functions. .

Just for the completeness of this section one example of how to create a new variable in R. Here a (fictional) year of birth is computed from the study date and the age of the individual subject. This also serves as an example of date arithmetic in R.

```
# Compute year of birth

library("lubridate")
studyDate <- as.Date("2014-06-01")
YearOfBirth <- year(studyDate  - years(data1$Age))
YearOfBirth
```

```
 [1] 1971 1963 1966 1972 1965 1967 1964 1953 1971 1963 1967 1969 1960 1962 1972
[16] 1970 1966 1951 1973 1963 1975 1974 1975 1976 1975 1973 1958 1958 1976 1957
[31] 1967 1966 1953 1965 1962 1959 1969 1972 1965 1964
```

This example uses the 'years()' function of the extension package 'lubridate' to simply subtract the years (of age) from the study date. It is assumed that the 'lubridate' package is already installed. To use a package that is not part of the base functionality of R, it has to be explicitly loaded with the 'library()' function before usage.

The result of this date arithmetic is a numerical vector with one value for each record of the original data frame. To add this vector to an existing data frame, it is appended as new variable by the 'cbind()' function of R. This is similar to match merging data with the 'merge' option in the SAS data step.

```
data2 <- cbind(data1, YearOfBirth)
```

DATA EXPLORATION

After removing or replacing missing values, it is time to look at the data itself. Somewhat similar to PROC MEANS in SAS the 'summary()' function of R creates a standardized overview of the values for all variables. It calculates the minimum, first quartile, arithmetic mean, median, third quartile and maximum values for numerical variables and counts the occurrence of factor levels for factor variables.

```
summary(data1)
```

```
    Subject       Treatment  DBP.Baseline     DBP.Month1      DBP.Month2
 Min.   : 1.00    : 0      Min.   :114.0   Min.   :111.0   Min.   :100.0
 1st Qu.:10.75   A:20      1st Qu.:115.0   1st Qu.:113.0   1st Qu.:112.0
 Median :20.50   B:20      Median :116.5   Median :115.0   Median :113.0
 Mean   :20.50             Mean   :116.7   Mean   :114.3   Mean   :112.4
 3rd Qu.:30.25             3rd Qu.:118.0   3rd Qu.:115.0   3rd Qu.:113.0
 Max.   :40.00             Max.   :121.0   Max.   :119.0   Max.   :118.0

   DBP.Month3      DBP.Month4         Age         Sex
 Min.   :102.0   Min.   : 97.0   Min.   :38.00    : 0
 1st Qu.:106.8   1st Qu.:101.8   1st Qu.:42.00   F:18
 Median :109.0   Median :106.5   Median :48.00   M:22
 Mean   :109.3   Mean   :106.7   Mean   :47.83
 3rd Qu.:113.2   3rd Qu.:112.0   3rd Qu.:51.25
 Max.   :117.0   Max.   :115.0   Max.   :63.00
```

In R, a large number of statistical functions exist, e.g. 'mean()', 'median()', 'quantile()', 'min()', 'max()' etc. If there are still remaining missing values in the data the function argument 'na.rm' specifies how to handle missing values: By setting na.rm = TRUE a listwise deletion of missing data takes place, similar to the NOMISS option in SAS procedures.

```
mean(data1$Age, na.rm=TRUE)
```

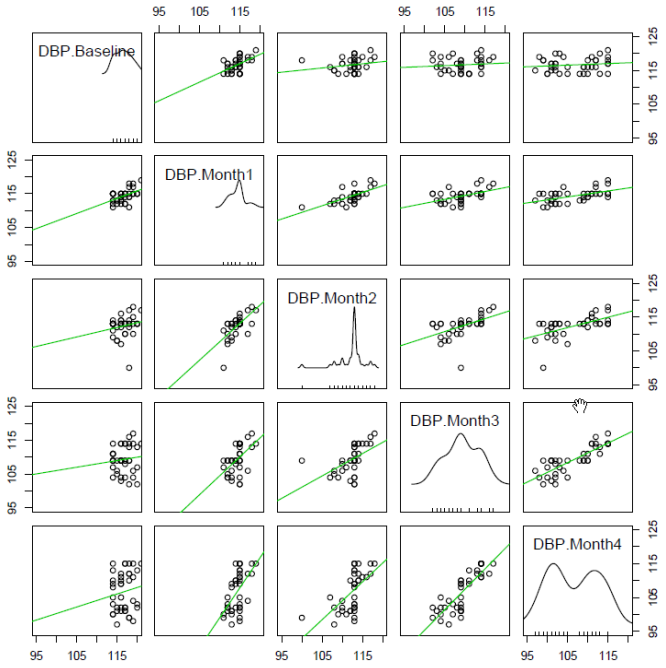
One of the great strengths of R is its graphics capabilities. It is good practice to have a look at some plots for a preliminary exploration. Scatterplot matrices are especially useful in this respect.

In the following example, all blood pressure variables are plotted with an enhanced scatterplot function of the 'car' package.

```
library("car")
```

```
data2 <- data1[,c("Treatment","DBP.Baseline","DBP.Month1","DBP.Month2",
       "DBP.Month3","DBP.Month4")]
```

```
scatterplotMatrix(data2[,-1], smoother=FALSE, xlim=c(95,120), ylim=c(95,125))
```



In addition to the bivariate scatterplots, the principal diagonal of the matrix contains univariate variable distributions. The green lines in the scatterplots are regression lines.

ASSESSING DIFFERENCES IN BLOOD PRESSURE BETWEEN TREATMENT GROUPS

The scatterplot matrix above shows that the blood pressure value of some subjects decreases over time and the final distribution of blood pressure values is bimodal in the last treatment month (DBP.Month4).

The question is now: Is there a reduction of blood pressure of the treatment group, and if so: Can this it be interpreted as a result of the treatment? First, let's compute the difference between the blood pressure before and after treatment and have a look at the mean difference for each group:

```
diff <- data1$DBP.Month4 - data1$DBP.Baseline
```

```
by(diff, data1$Treatment, mean)
```

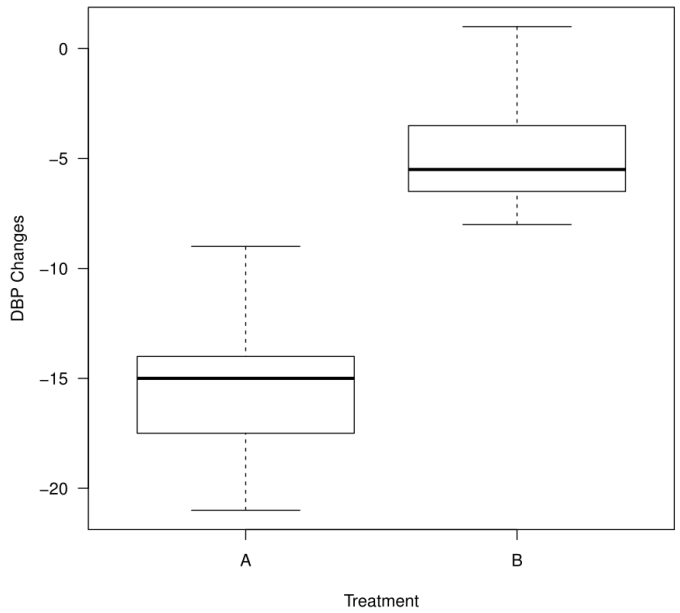
```
data1$Treatment: A
[1] -15.2
------------------------------------------------------------
data1$Treatment: B
[1] -4.8
```

Separate mean values were computed for each treatment group using the 'by()' function of R. This function splits the data by the treatment factor levels and applies the 'mean()' function given as third argument to every subset of the data. This is similar to 'by' processing in some of the PROC statements of SAS.

The new drug treatment „A" seems to be more effective than the „Placebo" treatment „B", since on average the decrease of diastolic blood pressure for drug A is larger than for „B" (-15.2 vs. - 4.8 mg Hg).

This effect is also visible in the boxplots created with R's 'boxplot()' function, or in SAS with PROC BOXPLOT.

```
boxplot(diff ~ Treatment, data2, xlab="Treatment", ylab="DBP Changes", las=1)
```



According to the boxplot, the data appears to be reasonably symmetric for the both treatment levels, implying that there are no obvious outliers.

This difference should be formally tested to assess whether it is statistically significant. This can be done with the function 't.test()' in R or with PROC TTEST in SAS. Let us assume equal variances and look at the t-test function call and its results:

```
ttResult <- t.test(diff ~ Treatment, data1, var.equal=TRUE)
```

The expression "diff ~ Treatment" is a formula in R and means "diff by Treatment". 'Treatment' has to be a factor variable with two levels. The function argument 'var.equal = TRUE' computes the t test for equal variances. The result is assigned to the object "ttResult".

```
ttResult
```

```
        Two Sample t-test

data:  diff by Treatment
t = -12.1504, df = 38, p-value = 1.169e-14
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -12.132758  -8.667242
sample estimates:
mean in group A mean in group B
          -15.2            -4.8
```

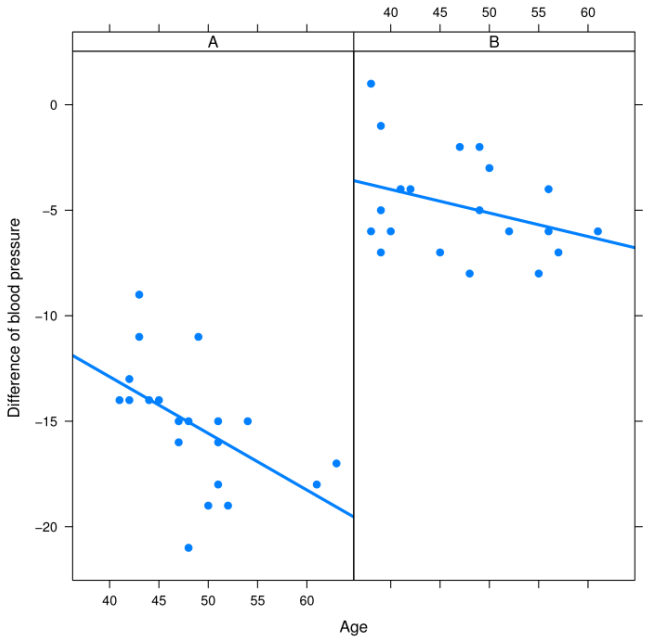
The t-statistic has a value of -12.15 with 38 degrees of freedom, which gives a very small p-value. That indicates that the difference between the two treatment groups (in terms of decrease of blood pressure) is strongly significant.

We could also question the assumption of equal variances in this test. The results with the option 'var.equal = FALSE" are very close to the results of the t-test above. An F-test for equality of variances also shows insufficient evidence to reject the null hypothesis of equal variances.

Further investigation indicates that there is a strong relationship of the blood pressure with both the Treatment and Age variable. The following trellis / lattice plot shows this relationship. SAS creates this kind of plots with PROC SGPANEL.

```
library(lattice)
```

```
print(xyplot(diff ~ Age | Treatment
   , data=data1
   , xlab="Age"
   , strip=strip.custom(bg="white")
   , ylab="Difference of blood pressure "
   , lwd=3, cex=1.3, pch=20, type=c("p", "r"))
)
```



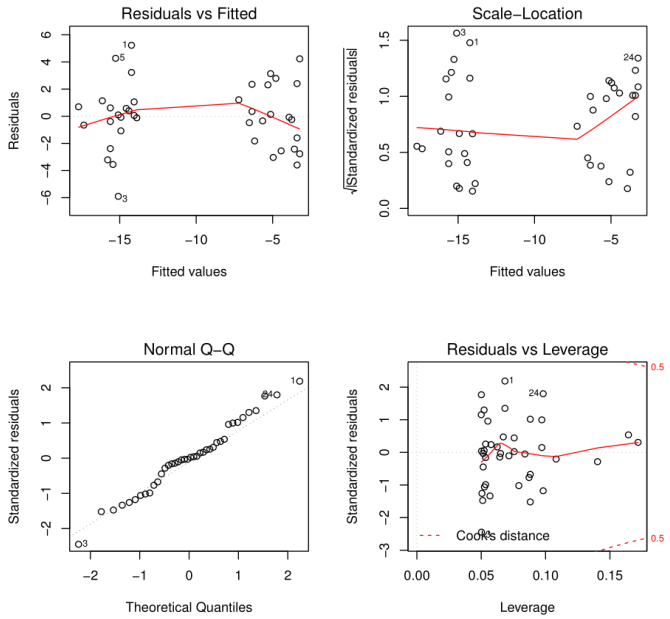
The relationship is statistically significant at the 0.001 level for Treatment and at the 0.01 level for Age. The following R example shows how to fit a linear model to the data with the 'lm()' function, then compute the analysis of variance tables from this model with the 'anova()' function.

```
# First fit a linear model and create a model object …
lmResult = lm(diff ~ Treatment + Age, data=data1)
# … then compute the analysis of variance tables from it.
anova(lmResult)

Analysis of Variance Table

Response: diff
          Df  Sum Sq Mean Sq  F value    Pr(>F)
Treatment  1 1081.60 1081.60 176.0395 1.228e-15 ***
Age        1   51.07   51.07   8.3119  0.006525 **
Residuals 37  227.33    6.14
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In SAS linear models are computed with PROC GLM or PROC REG, ANOVA models with PROC GLM, PROC ANOVA and PROC MIXED.

Diagnostic plots for the linear model are also available with the 'plot()' function. 'plot()' is a generic function: The actual plot type depends on the object type in the function argument. Plotting a linear model result creates four plots. The 'layout()' function positions the four plots in two rows and two columns.

```
layout(matrix(1:4, nrow=2))
plot(lmResult)
```



SAS does this with ODS Graphics and the PLOTS option in PROC GLM.

CREATING A PDF REPORT

Now you may wish to create a report from all data management and analysis steps. Producing a dynamic PDF report is quite simple with the 'Sweave()' function from the base R installation or with the 'knitr' package. Both solutions embed R code into latex documents; 'knitr' also supports other markup languages and output formats. Whenever the data or the code has changed, a new PDF is created by re-running the report generator. First, the R code is executed, and then the results (tables, plots etc.) are embedded in the document template and converted to a PDF file. This is similar to ODS and PROC DOCUMENT in SAS.

In addition, working with latex allows for the production of high quality tables with the 'xtables' package. This is an example of a formatted table in a PDF document created from the ANOVA result computed earlier:

Table 3: ANOVA table for diastolic blood pressure data

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Treatment	1	1081.60	1081.60	176.04	0.0000
Age	1	51.07	51.07	8.31	0.0065
Residuals	37	227.33	6.14		

CONCLUSION

The examples presented above show that R is different to SAS and the concept of the data step does not exist in R. If you do not expect R to be like SAS, R can be quite friendly, even for beginners: It is free, it is simple to install, good introductory books have been published and even easy to use editors and development environments are freely available.

You find several references to R-related books, web sites and papers in the "recommended reading" section below.

RECOMMENDED READING

Books and papers

- A very good introductory book for R is "R in action", Second edition (Early Access Edition, 2013) by Robert Kabacoff.
- For a comparison between SAS and R with a large number of side-by-side code examples please have a look at "SAS and R", Second Edition (2014) by Ken Kleinman and Nicholas J. Horton.
- Din Chen and Karl E. Peace cover the topic of clinical trial data analysis in their book "Clinical Trial Data Analysis" (2010).

How and where to find R packages

- Finding the right R package is an art in itself. Please have a look at the CRAN Task Views at the CRAN website: http://cran.r-project.org/web/views/, especially the Task View “Clinical Trial Design, Monitoring, and Analysis” and “Pharmacokinetics”

How to keep up to date

- When something interesting or important happens in the “R universe”, a blogger will report on it. This will eventually show up on the R-bloggers web site at http://www.r-bloggers.com/ .

Background and advanced topics

- A very interesting read from the main creator of the S language is the book “Software for Data Analysis: Programming with R” (2008) by John Chambers.
- “R Language Definition”, URL: http://cran.r-project.org/doc/manuals/r-release/R-lang.pdf
- Edwin de Jonge and Mark von der Loo published a paper on data cleaning: “An introduction to data cleaning with R” (2013), URL http://cran.r-project.org/doc/contrib/de_Jonge+van_der_Loo-Introduction_to_data_cleaning_with_R.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Friedrich Schuster
HMS Analytical Software GmbH
Rohrbacher Strasse 26
Heidelberg / 69115
Work Phone: +49 6221 6051142
Fax: +49 6221 6051742
Email: friedrich.schuster@analytical-software.de
Web: http://www.analytical-software.de/en/start/