Preamble

- An excellent website outlining data operations and basic data analysis with R can be found at http://www.statmethods.net. It makes frequent references to the book by Kabacoff.
- Explore the R-Project website: https://www.r-project.org/
- Explore Microsoft's Open R website: http://mran.microsoft.com/
- Explore RStudio's website: https://www.rstudio.com/

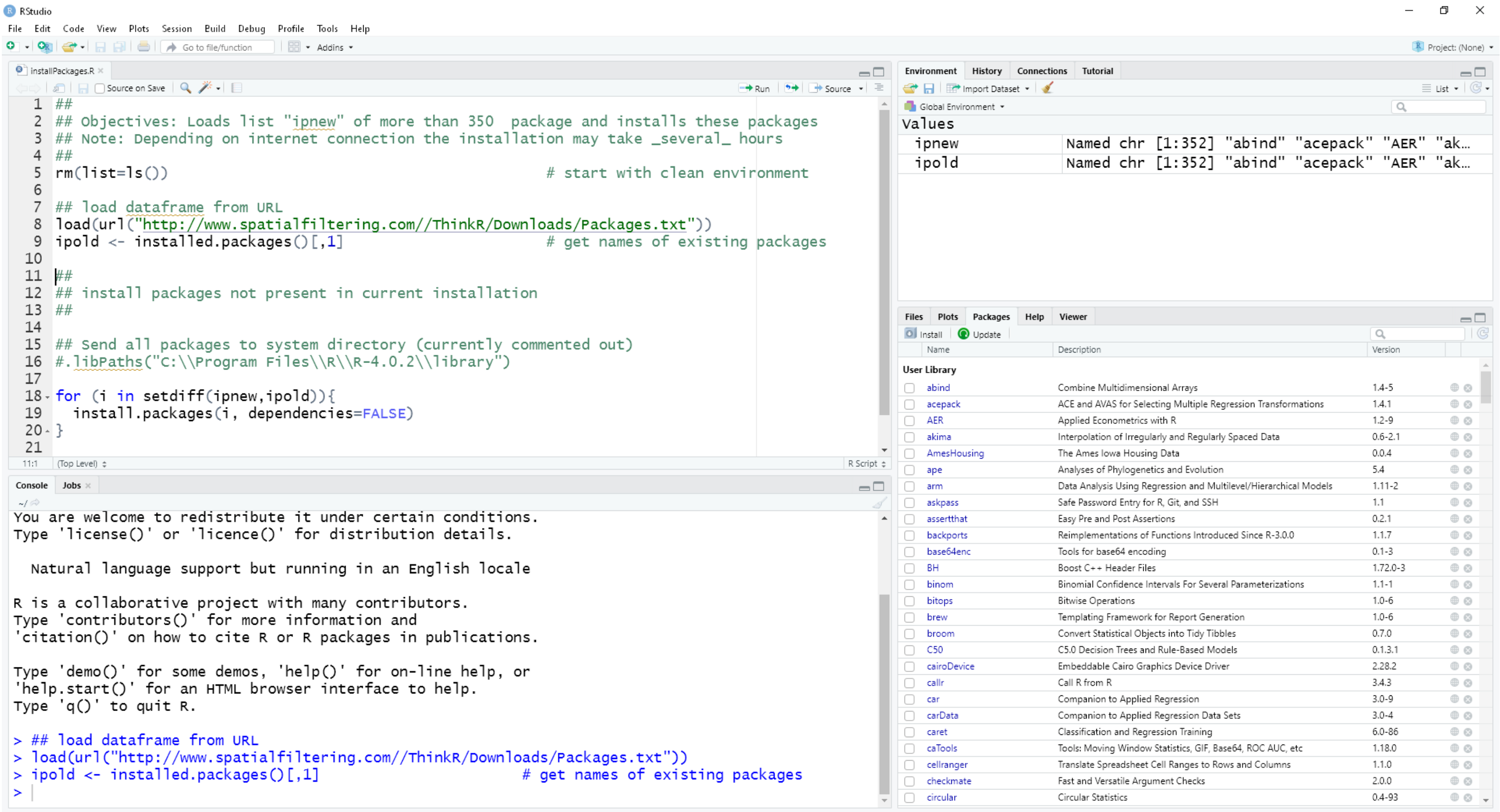
Reproducible Research

- R is ***not*** a menu driven computing environment.
- Rather it mainly uses commands (i.e., functions), which can be pooled together into scripts.
- The disadvantage of this approach is a higher learning effort to know the commands for your specific task.
- A substantial advantage of this approach is that you or anyone else at any time can exactly reproduce and check what you were doing.

  Thus R is strongly embedded in the ***academic paradigm of*** **Reproducible Research**.
- Furthermore, by analyzing your scripts instructors and teaching assistants can efficiently help you.

The RStudio Environment

- Discuss the RStudio interface structure: **Editor – Console – Environment/History/Connections – Files/Plots/Packages/Help/Viewer**



```r
##
## Objectives: Loads list "ipnew" of more than 350  package and installs these packages
## Note: Depending on internet connection the installation may take _several_ hours
##
rm(list=ls())                                       # start with clean environment

## load dataframe from URL
load(url("http://www.spatialfiltering.com//ThinkR/Downloads/Packages.txt"))
ipold <- installed.packages()[,1]                   # get names of existing packages

##
## install packages not present in current installation
##

## Send all packages to system directory (currently commented out)
#.libPaths("C:\\Program Files\\R\\R-4.0.2\\library")

for (i in setdiff(ipnew,ipold)){
  install.packages(i, dependencies=FALSE)
}

```

```
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> ## load dataframe from URL
> load(url("http://www.spatialfiltering.com//ThinkR/Downloads/Packages.txt"))
> ipold <- installed.packages()[,1]                   # get names of existing packages
> 
```

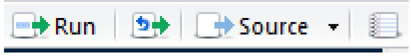
Values

ipnew	Named chr [1:352] "abind" "acepack" "AER" "ak…
ipold	Named chr [1:352] "abind" "acepack" "AER" "ak…

	Name	Description	Version
	User Library		
□	abind	Combine Multidimensional Arrays	1.4-5
□	acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
□	AER	Applied Econometrics with R	1.2-9
□	akima	Interpolation of Irregularly and Regularly Spaced Data	0.6-2.1
□	AmesHousing	The Ames Iowa Housing Data	0.0.4
□	ape	Analyses of Phylogenetics and Evolution	5.4
□	arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.11-2
□	askpass	Safe Password Entry for R, Git, and SSH	1.1
□	assertthat	Easy Pre and Post Assertions	0.2.1
□	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.7
□	base64enc	Tools for base64 encoding	0.1-3
□	BH	Boost C++ Header Files	1.72.0-3
□	binom	Binomial Confidence Intervals For Several Parameterizations	1.1-1
□	bitops	Bitwise Operations	1.0-6
□	brew	Templating Framework for Report Generation	1.0-6
□	broom	Convert Statistical Objects into Tidy Tibbles	0.7.0
□	C50	C5.0 Decision Trees and Rule-Based Models	0.1.3.1
□	cairoDevice	Embeddable Cairo Graphics Device Driver	2.28.2
□	callr	Call R from R	3.4.3
□	car	Companion to Applied Regression	3.0-9
□	carData	Companion to Applied Regression Data Sets	3.0-4
□	caret	Classification and Regression Training	6.0-86
□	caTools	Tools: Moving Window Statistics, GIF, Base64, ROC AUC, etc	1.18.0
□	cellranger	Translate Spreadsheet Cell Ranges to Rows and Columns	1.1.0
□	checkmate	Fast and Versatile Argument Checks	2.0.0
□	circular	Circular Statistics	0.4-93

- Explore RStudio’s Tools and Help tab and R’s help system.
- Concept of environment, history file and script. The associate file extensions are:
  - **`*.RData`** or **`*.rda`**: Copy of the workspace with all its data objects and custom user functions.

  - ○ **`*.Rhistory`** contains all commands that have been issued during a session at the command prompt `>`.
  - ○ **`*.R`** is a file that contains scripts, which can be a set of basic R data analysis commands, individual functions, or an elaborate program
- Get and change the ***working directory*** were your scripts and workspace are stored and searched by default.

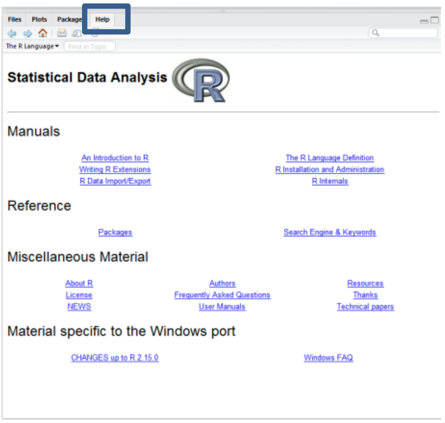
  Enter the command **`getwd()`** into the editor window and select “Run” from 
.
  Your current working directory is displayed in the **CONSOLE** below
  Each command in R ends with parentheses, e.g., **`getwd()`**, which may include optional parameters. Even if no options are specified a function always ends with parentheses **`( )`**

  <u>Comments:</u>

  - ○ The ***string*** **`"C:/Users/Michael/Documents"`** is the path to the working directory. In R strings are always enclosed by quotation marks, i.e, **`"…"`**.
    The **WINDOWS** convention is to separate sub-directories by a backward slash **`\`**.
    However, R uses the forward slash **`/`** or alternatively a double backward slash **`\\`**.
    In R the single backslash **`\`** is reserved to start an ***escape*** character, for instance, **`\n`** becomes a line break and carriage return in text output, e.g., **`> cat("First line\n Second line")`** is shown in the **CONSOLE** on two lines.
  - ○ The character **`>`** in the **CONSOLE** window is the command prompt which indicates that R is ready to receive new commands. It shows up when R completed executing a script.
  - ○ The **ESC** key or pressing STOP in the **CONSOLE** window – available while a script is executing – can terminate the ongoing execution of a script.

- Receiving help at the command prompt **>** in the **CONSOLE**:
  - ○ **`> help(`*`FunctionName`*`)`** or short **`> ?`*`FunctionName`***

    This searches all ***active*** libraries (libraries currently linked to your session) for the help.
  - ○ Or through the **HTML**-help menu system:



Statistical Data Analysis

Manuals

An Introduction to R
Writing R Extensions
R Data Import/Export
The R Language Definition
R Installation and Administration
R Internals

Reference

Packages
Search Engine & Keywords

Miscellaneous Material

About R
License
NEWS
Authors
Frequently Asked Questions
User Manuals
Resources
Thanks
Technical papers

Material specific to the Windows port

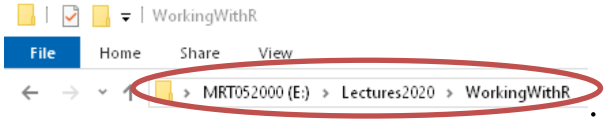
CHANGES up to R 2.15.0
Windows FAQ

  - ○ Fuzzy help can be obtained with the double question mark **`> ??`*`PartName`***. This searches all installed libraries for the help on functions, data or vignettes containing the string “PartName”.

Basic R Mechanics

Make your Working Directory

- My suggestion is that you setup a specific directory on your hard- or jump-drive for this course, e.g.,



MRT052000 (E:) > Lectures2020 > WorkingWithR

.

You store your scripts, data etc. in this directory or its subdirectories.

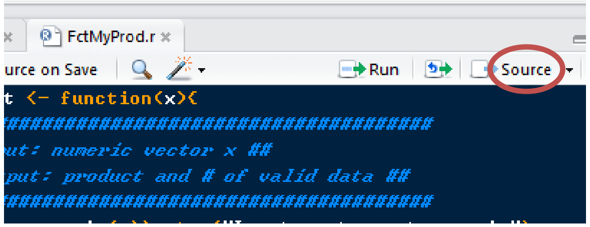
- Then set your working directory to this location with the command

```
> setwd("E:\\Lectures2020\\WorkingWithR").
```

- Notice the double backward slash \\ here as substitute for the single forward slash **/** and the use of the quotation **"…"** to enclose the directory string.

Interacting with the R-Console

- Collections of commands (or programs) can be stored in external `*.R` script-files (see **FILE** menu)
- Single commands in the script editor can be run from the command prompt or with **RUN** for a highlighted line in a script.
- To run a script (all lines in your editor) use the **SOURCE** button:



```
t <- function(x){
######################################
ut: numeric vector x ##
put: product and # of valid data ##
######################################
```

- In the **CONSOLE** window the arrow **UP** and **DOWN** keys scrolls through the history of previously issued R commands.
- Previously issued commands can be edited by moving with the arrow keys the edit prompt to a particular location of your command line.
- While being in the **CONSOLE** window, commands with their list of parameters can be broken over several lines. Then the continuing prompt "**+**" will be displayed ***automatically*** at the beginning of a new line until the command is completed.

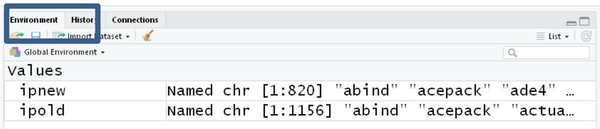
- To clean a cluttered **Console** window use the key combination **Ctrl-L** or use the broom icon 🧹 .

Variables

- The assignment operator to a variable is the backward arrow "**`<-`**", i.e., it requires two key-strokes, **`> my.name <- "Michael"`** (The variable **`my.name`** will now show up in the **Environment**) with the content **`"Michael"`**)

  Note: The backward arrow **`<-`** is preferred over the equal sign **`=`** , which is reserved for parameter assignments in function calls.
- ***Variable names***:
  - Names must begin with a letter and can consist of an alpha/numeric combination of letters including the period "**`.`**" and/or the underscore "**`_`**"
  - Note: Specific characters and keywords such "**`$`**" "**`@`**" "**`&`**" and "**`%`**" or "**`+`**" "**`-`**" "**`*`**" "**`/`**" and "**`^`**" **cannot** be used because they have special meanings. For a full list see **`> ?Reserved`**
  - Warning: Variable and function names in R are ***case sensitive***, e.g., ***`my.Var`*** and ***`my.var`*** are different objects (⇒ this can lead to typos while writing a script. These are extremely difficult to spot)
  - Tip: Name variables properly so an external reader or you, after a few weeks have passed, can understand what you were doing
  - Use the dot to structure associated variable names, e.g., ***`sales.plano`*** and ***`sales.dallas`***, or the ***camelback*** convention ***`salesPlano`*** and ***`salesDallas`***
  - Just for experts: The document **GOOGLE-R-STYLE.PDF**. suggests professional naming and typesetting conventions of your R code.

- Any ***function*** or ***data structure*** that is defined during a R-session becomes an object in the **ENVIRONMENT**



```
Values
  ipnew          Named chr [1:820] "abind" "acepack" "ade4" …
  ipold          Named chr [1:1156] "abind" "acepack" "actua…
```

- These can be removed from the **ENVIRONMENT** with the remove function

  ```
  > rm(ipold)
  ```

- To clean the everything from the environment use the ***nested*** commands `> rm(list=ls())` or the broom icom 🧹 in the **ENVIRONMENT** menu bar.
- Warning: if you happen to name a variable or function identically to an existing R object, which already exists in the ***search path*** of your session, therefore, that object will be masked and is no longer directly accessible:

  ```
  > pi              # gives the system constant 3.141593
  > pi <- 2.71      # this masks the system constant pi
  > base::pi        # pi still be found in its library base. Note the "::"
  > rm(pi)          # removes user's pi and makes the constant pi available
  ```

- Some hard-wired values in R are:
  - ***Logical*** values **`T`** and **`F`** (alternatively **`TRUE`** and **`FALSE`** can be used)
  - An object ***without content*** has the value **`NULL`**
  - ***Impossible*** operations, such as **`log(-1)`**, lead to a *not-a-number* **`NaN`**.
  - ***Missing*** numbers have the value **`NA`** which stands for not available.
  - Some ***predefined*** numbers are infinity **`Inf`** and **`pi`** (that is, $\pi = 3.141593$)

Data Representation in R

Data-Sets

- For statistical analyses data-sets are usually arranged in rectangular data-frames and imported from external files (such as **SPSS**, **STATA, EXCEL** or **DBASE**) or embedded in workspaces (with the extension **`*.RData`**) of libraries.

  For instance, to read an SPSS file use:

  ```
  > library(foreign)  # makes import functions available during a session
  > setwd("E:\\Lectures2020\\WorkingWithR")
  > MyPower <- read.spss("DallasTempPower.sav", to.data.frame=T)
  ```

  or more compactly skipping the attachment of library **FOREIGN**:

  ```
  > MyPower <- foreign::read.spss("DallasTempPower.sav", to.data.frame=T)
  ```

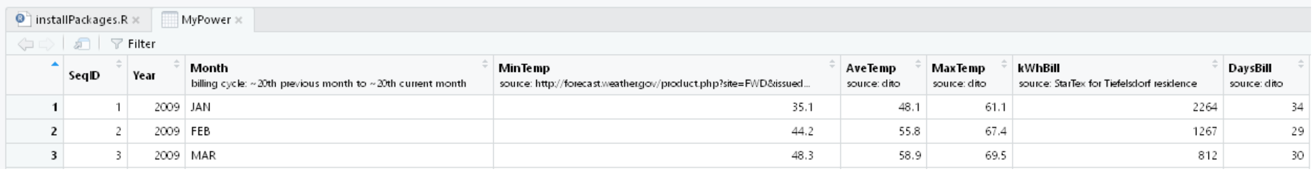
- Some data file types can also be imported using **FILE ▶ IMPORT DATASET** or with Import Dataset ▾ in the **ENVIRONMENT** window. Note, however, using the menu violates the reproducible paradigm.
- Notice that the object **`MyPower`** is added as data-frame to your **ENVIRONMENT**.
- To calculate a new variable and assign it to the data-frame use the syntax **`df$NewVar`**, e.g.

  ```
  > MyPower$DiffTemp <- MyPower$MaxTemp - MyPower$MinTemp
  ```

- Check the class of an object use

  ```
  > class(MyPower)
  [1] "data.frame"
  ```

- To preview the data-frame double click on it in the **ENVIRONMENT**.



	SeqID	Year	Month	MinTemp	AveTemp	MaxTemp	kWhBill	DaysBill
			billing cycle: ~20th previous month to ~20th current month	source: http://forecast.weather.gov/product.php?site=FWD&issued...	source: dito	source: dito	source: StarTex for Tiefelsdorf residence	source: dito
1	1	2009	JAN	35.1	48.1	61.1	2264	34
2	2	2009	FEB	44.2	55.8	67.4	1267	29
3	3	2009	MAR	48.3	58.9	69.5	812	30

- Each column in a data-frame has an associated elementary data type.
- Some R libraries also include their own data-frame. These can be opened by the command

```
> data("CPS1985", package="AER")
```

Elementary Data Types

- Within a data-frame the variables can be of an elementary R data types:
  - logical: **`FALSE`** or **`TRUE`** (also binary **`0`** or **`1`**), e.g., **`am.I.logical <- TRUE`**
  - character strings: always enclosed by single or double quotation marks, e.g., **`my.name <- "Michael"`**.
  - factors: Internally each value is stored with a specific integer number. Each integer has a descriptive label assigned to it, which is displayed.

    Notice: factor level values are not enclosed within quotation marks. For instance:

```
> MyPower$Month
 [1] JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC
[13] JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC
[25] JAN FEB MAR APR MAY JUN JUL AUG
Levels: JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC
```

Individual factor levels group the observations according to their specific factor level.
The syntax **MyPower$Month** addresses the factor variable **Month** in the data-frame **MyPower**.

- numeric: either integer, double precision or complex numbers, which are not used in this course.
  - True integer values can be enforced with the added symbol "**L**"

    ```
    > two <- 2L.
    > typeof(two)
    [1] "integer"
    ```

  - Integers, under specific circumstance, are treated as real numbers or may be ***coerced*** to real numbers

    ```
    > two.sq <- two * 2.0
    > typeof(two.sq)
    [1] "double"
    ```

  - Very small remainders after internal rounding operations are interpreted as zeros, such as

    ```
    > (sqrt(2))^2 - 2
    [1] 4.440892e-16.
    ```

    This discrepancy is due to ***rounding errors*** associated with floating point computations, i.e. taking the square root and squaring the remaining results. This problem applies to all software environments and computer chips dealing with floating point numbers.
    Floating point numbers with an infinite number of digits can only be ***digitally approximated*** up to a given depth due to the limited bits implemented in operating systems.
    E.g., while the constant $\pi = 3.14 \ldots$ has an infinite number of digits, only the first 16 can be numerically represented:

```
> options(digits=20)
> pi
[1] 3.1415926535897931
```

- Dates: Dates come in different formats. This course will ***not*** cover dates.

Basic Data Objects in R:

- scalar: an individual datum. All data objects are composed of a collection of scalars.
- vector: ***atomic data structure*** that collects more than one value of ***identical type***. Example:

  `> score <- c(23,53,45,30,53,60)` or

  `> catName <- c("Austin","Gretchen","Charlie")`

  where the function `c()` concatenates – combines –several scalars into a vector.

  `> length(score)` gives the number of elements in the vector.

  Note: if a printed vector stretches over several rows in the console the position of the first value in each row will be numbered by `[elementNumber]`

  Mixture of scalars with different data types will be ***coerced*** into characters

  ```
  > mixture <- c(1,2,3,T,pi,"A”)
  > mixture
  [1] "1"  "2"  "3"  "TRUE"  "3.14159265358979"  "A"
  ```
- matrix: two-dimensional arrangement of a set of vectors that are all of the ***same data type and length***.
- data frames: collection of vectors of the ***same length*** but perhaps ***different data types*** per column.
- list: collection of vectors of any type and length.

- To obtain information about a data object and to look at its structure use the function `str()`:

```
> str(MyPower)
'data.frame':    32 obs. of  8 variables:
 $ SeqID   : num  1 2 3 4 5 6 7 8 9 10 ...
 $ Year    : num  2009 2009 2009 2009 2009 ...
 $ Month   : Factor w/ 12 levels "JAN","FEB","MAR",...: 1 2 3 4 5 6 7...
 $ DaysBill: num  34 29 30 32 29 30 32 29 30 31 ...
 - attr(*, "codepage")= int 1252
```

Object Oriented Philosophy

- R is an object oriented data analysis language, which is centered around functions that are applied to objects.
- Functions may change their behavior – if properly programmed – in response to the class of their input argument.
- All data objects have an ***object class*** assigned to them. This may be fairly rudimentary, such as being a numeric vector, or highly advanced, such as the output from a regression model.
- All R commands are ***implemented as functions*** with opening and closing parentheses at the end of the function name.

  A function may not use [a] specific input arguments, e.g., the function `getwd()`,or [b] a selection of arguments, [c] some of these arguments may have default values and only need to be issued when they are overwritten, and [d] some arguments are positional (without a leading keyword) or are introduced by a specific keyword. For instance:

```
> result <- myFunc(pos1, pos2, keyword1=TRUE, keyword2="align")
```

where the object class of **`result`** is assigned by the function **`myFunc`** and the behavior of the function is defined by the object class of its first positional input argument **`pos1`**.

- Depending on the class of the input arguments, an object-specific function will be used.
  Example notice amorph behavior of the **`summary( )`** function:

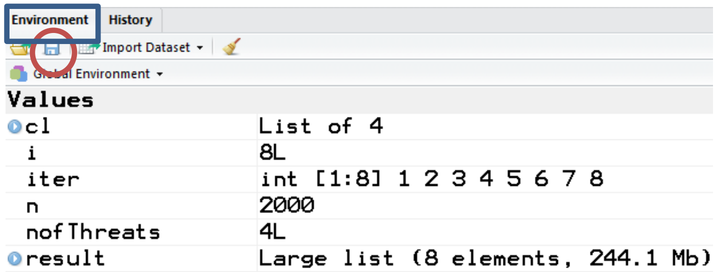
```
> class(MyPower)
> summary(MyPower)    # Get summary statistics of all variables
> ## run a regression model and save output into object MyReg
> MyReg <- lm(MaxTemp ~ MinTemp, data=MyPower)
> class(MyReg)
> summary(MyReg)      # view key results of the regression analysis
```

- If just the object name is entered at the command prompt its *default print method* will be used. For instance,

```
> str(MyReg)               # print list content of  MyReg on the console
```

Saving Analysis Results, Plots and Data

- Analysis Results and Plots: The easiest way of saving [a] **CONSOLE** output and [b] generated plots in the graphics window is to ***copy and past*** them into a graphically enhanced text editor such as **WORD**.
  Important: Any text output needs to be typeset in a ***fix pitch font*** such as **`Courier New`**.
  Otherwise the formatting of the output will be lost, e.g., columns of a matrix ***will not*** line up properly.
  Perhaps reduce the font size and line-spacing of imported output as well as switch to a landscape layout.
- Data: The collection of all variables, data-frames and functions, which were created during a session, can be saved for subsequent sessions in a workspace



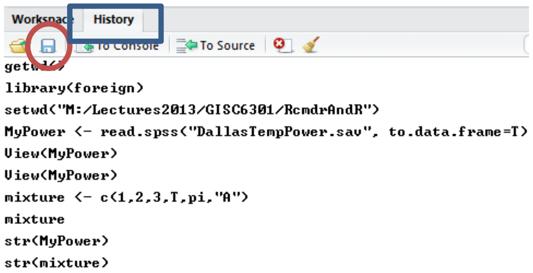
Values

cl	List of 4
i	8L
iter	int [1:8] 1 2 3 4 5 6 7 8
n	2000
nofThreats	4L
result	Large list (8 elements, 244.1 Mb)

Before saving the workspace non-desired data-objects, variables and functions should be dropped with the remove command:

> rm(objectName).

- Data-frames within the workspace can also be ***exported*** into different file formats (see the package **`foreign`** or connections to **SQL** servers).
- <u>Command History:</u> All commands, which were issued during session, can be saved into a history file



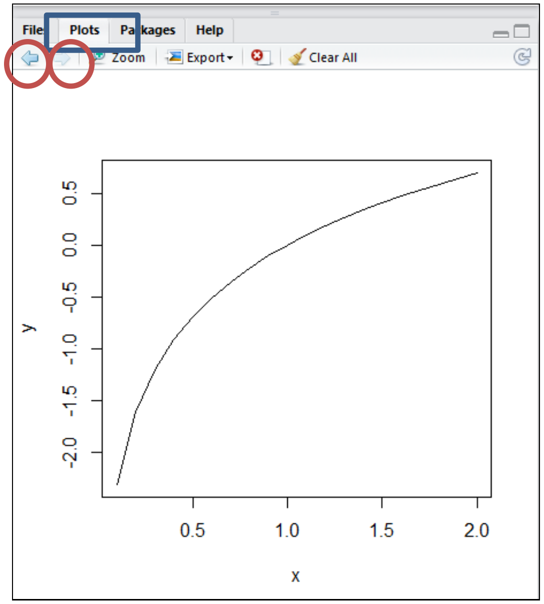
```
getwd()
library(foreign)
setwd("M:/Lectures2013/GISC6301/RcmdrAndR")
MyPower <- read.spss("DallasTempPower.sav", to.data.frame=T)
View(MyPower)
View(MyPower)
mixture <- c(1,2,3,T,pi,"A")
mixture
str(MyPower)
str(mixture)
```

Working with some R functions

- ***<u>Exercise:</u>***

```
> x <- seq(0.1 ,2,by=0.1)   # sequence of numbers: 0.1,0.2,…,2
> x                         # show numbers
> y <- log(x)              # calculate the natural logarithm
> plot(x,y)                # Plot y against x
> plot(y~x)                # same plot conceiving y as a function of x
> help(plot)               # Explore options
> plot(x,y,type="l")       # Connect points by lines
```

- Select in the Plots window scroll through your list of plots with the arrow icons:



- ***Exercise (cont.):***

```
> z <- rnorm(length(x))  # vector of standard normal random numbers
> mat <- cbind(x,y,z)    # merge vectors of same length into a matrix
> dim(mat)               # see the dimensions of the matrix
> summary(mat)             # get statistics of each matrix column
> class(mat)               # evaluate object type
[1] "matrix"
> df <- data.frame(x=x, xlog=y, rand=z) # build dataframe
> class(df)
[1] "data.frame"
```

Working with Data

Data-Frames

- Data-frames can pool several vectors of ***same length*** but potentially of ***different data-types*** together.
- Almost all statistical analysis functions are defined on data-frames.

```
> catName <- c("Austin","Gretchen","Charlie")       # character vector
> catAge <- c(9,10,20)                              # numeric vector
> my.data <- data.frame(Name= catName, Age= catAge) # Define data-frame
> my.data                                           # Show data-frame
      Name   Age
1   Austin     9
2 Gretchen    10
3  Charlie    20
```

- The variables **`catName`** and **`catAge`** are stored now in the data frame **`my.data`** under new names **`Name`** and **`Age`**
- To access individual variables in the data-frame several commands can be used

```
> my.data$Name
> my.data["Name"]
> my.data[1]
> with(my.data, Name)
> my.data[ , "Name"]
> my.data[ , 1]
```

List Objects

- Note: R functions can only return one data object. However, different data object can be bundled into a list and returned by the function.
- List objects allow to link data objects of ***different types*** and ***different length*** together into a container:

```
> A <- matrix(c(1,2,3,4,5,6), nrow=2, ncol=3)      # 2 x 3 matrix
> my.list <- list(name = catName, age = catAge, mat = A)
> my.list
$name
[1] "Austin"   "Gretchen"    "Charlie"
$age
[1]  9  10  20

$mat
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

- Individual objects of the list can be addressed either by

```
> my.list$name
[1] "Austin"   "Gretchen"    "Charlie"
> my.list[[1]]
[1] "Austin"   "Gretchen"    "Charlie"
```

- Individual elements of an object in a list can be addressed by

```
> my.list$name[1]
[1] "Austin"
```

- To delete an object in a list assign the **`NULL`** value:

```
> my.list$mat <- NULL
```

- To get information about an object use the attributes function:

```
> attributes(my.list)
$names
[1] "name" "age"
```

- Remove the objects **`my.list`** and the matrix **`A`** from the workspace

```
> rm(my.list,A)
```

Matrix Objects

- Matrices can store vectors of ***same length*** and ***same data-type*** in an rectangular arrangement
- To generate a matrix:
  - Vector of 12 elements:

    ```
    > b <- c(10,20,30,40,15,25,35,45,1,2,3,4)
    ```

  - Place elements into 4x3 matrix:

    ```
    > mat <- matrix(b, nrow=4, ncol=3)
    ```

- One element at location row and col **`mat[row,col]`**
  A sequence of values **`mat[1:2,]`** (here the first and second row)
  Exclusion of elements **`mat[-1,]`** (here the first row)
- One row **`mat[1,]`** or one column **`mat[,2]`**
- Also logical operations are permitted (here the first and second column):

  ```
  > select <- c(T,T,F)
  > mat[,select]
  ```