

# Beginner Friendly Introduction to R

October, 2017



# What is R

- **R is a system for statistical computing and graphics, includes:**
  - R language to script math operations
  - R environment to run R scripts
  - RServe – an interface for analytical applications
- **R is interpreted language**
  - You do not have to write a program to run it
  - Every line is interpreted on the fly
- **R is free under GNU General Public License**
- **R is maintained by volunteers at <https://www.r-project.org>**
- **R is well developed, well documented, and extensively used around the world**
  - Over 2000 extension packages expanding R functionality

# RStudio

- **RStudio is a development environment for R**
- **Created by a company called RStudio**
  - **A member of R community**
- **One can develop directly in R, but RStudio is more productive**
- **Every DSVD user will have his/her own RStudio Desktop**
- **RStudio closely resembles Eclipse**
- **Rstudio requires R**

# Working with RStudio in DSVD

- **Login into Horizon Client**
- **Open GSA Pool 6**
- **Open All Programs**
- **Open RStudio folder (not R folder!)**
- **Open Rstudio**
- **Connect to/upload your data**
- **Develop your R script**

# R Basic Concepts

- **Workspace**

The workspace is your current R working environment and includes any user-defined objects, e.g. variables and functions

- **R Session**

Time you work in R. At the end of an R session, the user can save an image of the current workspace and use it later

- **Data Types**

Numeric, integer, character, logical

- **Variables**

Scalars, vectors, matrices, data frames, and lists

- **Commands and Operators**

- **Functions**

A piece of code written to carry out a specific task

- **Comments**

# not executable lines

- **Extension Packages**

Additional functionality

- **Help**



# RStudio Interface

The screenshot shows the RStudio interface with several annotations in red text and red circles highlighting specific areas:

- Write R Script**: Points to the main script editor area on the left.
- Environment History**: A red circle highlights the 'Environment' and 'History' tabs at the top of the right-hand pane.
- Switch tabs to see**: A red circle highlights the 'Files', 'Plots', 'Packages', 'Help', and 'Viewer' tabs at the bottom of the right-hand pane.
- Commands**: A red circle highlights the 'Commands' tab in the 'Environment' pane.
- Variables**: A red circle highlights the 'Variables' tab in the 'Environment' pane.
- Create variables**: A red circle highlights the 'Create variables' tab in the 'Console' pane.
- Enter commands**: A red circle highlights the 'Enter commands' tab in the 'Console' pane.
- See Results**: A red circle highlights the 'See Results' tab in the 'Console' pane.
- Switch tabs to**: A red circle highlights the 'Switch tabs to' tab in the 'Console' pane.
- Import files**: A red circle highlights the 'Import files' tab in the 'Console' pane.
- Plot data**: A red circle highlights the 'Plot data' tab in the 'Console' pane.
- View installed packages**: A red circle highlights the 'View installed packages' tab in the 'Console' pane.

The RStudio interface includes a menu bar (File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help), a toolbar, a script editor (Untitled1), a console, and a right-hand pane with tabs for Environment, History, Files, Plots, Packages, Help, and Viewer. The console shows the R startup message and the R script editor shows the following code:

```
?reserved
A = [1,2,4,5,6]
a = 1,2,3,4
a = (1,2,3,4)
library(readxl)
Statistics_Exercises <-
sheet = "sheet6", col_t
"numeric", "blank"))
view(Statistics_Exercises,
view(Statistics_Exercises)
plot(Rand())
plot(Rand())
```

The console output shows the R startup message and the R script editor shows the following code:

```
?reserved
A = [1,2,4,5,6]
a = 1,2,3,4
a = (1,2,3,4)
library(readxl)
Statistics_Exercises <-
sheet = "sheet6", col_t
"numeric", "blank"))
view(Statistics_Exercises,
view(Statistics_Exercises)
plot(Rand())
plot(Rand())
```

# Data Types: Numeric and Integer

- **Numeric**

```
> k = 1
> k          # print the value of k
[1] 1
> class(k)   # print the class name of k
[1] "numeric"

> is.integer(k) # is k an integer?
[1] FALSE
```

- **Integer**

```
> y = as.integer(3)
> y          # print the value of y
[1] 3
> class(y)   # print the class name of y
[1] "integer"
> is.integer(y) # is y an integer?
[1] TRUE
```

# Data Types: Complex

- A **complex** value in R is defined via the pure imaginary value  $I = \sqrt{-1} = i$   

```
> z = 1 + 2i    # create a complex number  
> z            # print the value of z  
[1] 1+2i  
> class(z)      # print the class name of z  
[1] "complex"
```
- The following gives an error as  $-1$  is not a complex value  

```
> sqrt(-1)      # square root of -1  
[1] NaN  
Warning message:  
In sqrt(-1) : NaNs produced
```
- Instead, we have to use the complex value  $-1 + 0i$   

```
> sqrt(-1+0i)   # square root of -1+0i  
[1] 0+1i
```
- An alternative is to coerce  $-1$  into a complex value  

```
> sqrt(as.complex(-1))  
[1] 0+1i
```



# Data Types: Logical

- A **logical** value is often created via comparison between variables

```
> x = 1; y = 2 # sample values
> z = x > y    # is x larger than y?
> z           # print the logical value
[1] FALSE
> class(z)     # print the class name of z
[1] "logical"
```

- Standard logical operations are "&" (and), "|" (or), and "!" (negation)

```
> u = TRUE; v = FALSE
> u & v        # u AND v
[1] FALSE
> u | v        # u OR v
[1] TRUE
> !u           # negation of u
[1] FALSE
```

# Data Types: Character

- A **character** object is used to represent string values in R. We convert objects into character values with the `as.character()` function

```
> x = as.character(3.14)
> x          # print the character string
[1] "3.14"
> class(x)    # print the class name of x
[1] "character"
```

- Two character values can be concatenated with the `paste` function

```
> fname = "Joe"; lname = "Smith"
> paste(fname, lname)
[1] "Joe Smith"
```

# Data Types: Character (Cont.)

- It is often more convenient to create a readable string with the `sprintf` function, which has a C language syntax.  

```
> sprintf("%s has %d dollars", "Sam", 100)  
[1] "Sam has 100 dollars"
```
- To extract a substring, we apply the `substr` function. Here is an example showing how to extract the substring between the third and twelfth positions in a string.  

```
> substr("Mary has a little lamb.", start=3, stop=12)  
[1] "ry has a l"
```
- To replace the first occurrence of the word "little" by another word "big" in the string, we apply the `sub` function.  

```
> sub("little", "big", "Mary has a little lamb.")  
[1] "Mary has a big lamb."
```

# Variables: Vector

- A **vector** is a sequence of data elements of the same data type

- A vector containing three numeric values 2, 3 and 5

```
> c(2, 3, 5)
```

```
[1] 2 3 5
```

- A vector of logical values

```
> c(TRUE, FALSE, TRUE, FALSE, FALSE)
```

```
[1] TRUE FALSE TRUE FALSE FALSE
```

- A vector of character strings

```
> c("aa", "bb", "cc", "dd", "ee")
```

```
[1] "aa" "bb" "cc" "dd" "ee"
```

- The number of members in a vector is given by the length function

```
> length(c("aa", "bb", "cc", "dd", "ee"))
```

```
[1] 5
```

# Variables: Matrix

- A **matrix** is a collection of data elements arranged in a two-dimensional rectangular layout. The following is an example of a matrix with 2 rows and 3 columns

$$A = \begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

- We reproduce a memory representation of the matrix in R with the matrix function. The data elements must be of the same basic type

```
> A = matrix(  
+ c(2, 4, 3, 1, 5, 7),    # the data elements  
+ nrow=2,                 # number of rows  
+ ncol=3,                 # number of columns  
+ byrow = TRUE)          # fill matrix by rows
```

```
> A                        # print the matrix
```

```
  [,1] [,2] [,3]  
[1,]  2  4  3  
[2,]  1  5  7
```



# Variables: Matrix (Cont.)

- We reproduce a memory representation of the matrix in R with the matrix function. The data elements must be of the same basic type.

```
> A = matrix(  
+ c(2, 4, 3, 1, 5, 7), # the data elements  
+ nrow=2,              # number of rows  
+ ncol=3,              # number of columns  
+ byrow = TRUE)       # fill matrix by rows
```

```
> A                      # print the matrix  
      [,1] [,2] [,3]  
[1,]  2   4   3  
[2,]  1   5   7
```

- An element at the  $m^{th}$  row,  $n^{th}$  column of A can be accessed by the expression A[m, n].

```
> A[2, 3]    # element at 2nd row, 3rd column  
[1] 7
```

- The entire  $m^{th}$  row A can be extracted as A[m, ].

```
> A[2, ]     # the 2nd row  
[1] 1 5 7
```

# Variables: Matrix (Cont. 2)

- Similarly, the entire  $n^{th}$  column A can be extracted as A[,n].

```
> A[,3]      # the 3rd column
[1] 3 7
```

- We can also extract more than one rows or columns at a time.

```
> A[,c(1,3)] # the 1st and 3rd columns
      [,1] [,2]
[1,]    2    3
[2,]    1    7
```

- If we assign names to the rows and columns of the matrix, than we can access the elements by names.

```
> dimnames(A) = list(
+   c("row1", "row2"),      # row names
+   c("col1", "col2", "col3")) # column names
```

```
> A      # print A
      col1 col2 col3
row1    2    4    3
row2    1    5    7
```

```
> A["row2", "col3"] # element at 2nd row, 3rd column
[1] 7
```

# Variables: List

- A **list** is a generic vector containing other objects.
  - For example, the following variable x is a list containing copies of three vectors n, s, b, and a numeric value 3.
- ```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc", "dd", "ee")
> b = c(TRUE, FALSE, TRUE, FALSE, FALSE)
> x = list(n, s, b, 3) # x contains copies of n, s, b
```
- **List Slicing:** retrieve a list slice with the *single square bracket* "[" operator. The following is a slice containing the second member of x, which is a copy of s.

```
> x[2]
[[1]]
[1] "aa" "bb" "cc" "dd" "ee"
```

- With an index vector, we can retrieve a slice with multiple members. Here a slice containing the second and fourth members of x.

```
> x[c(2, 4)]
[[1]]
[1] "aa" "bb" "cc" "dd" "ee"
[[2]]
[1] 3
```

# Variables: List (Cont.)

- To reference a list member directly, we have to use the *double square bracket* "[[]]" operator. The following object `x[[2]]` is the second member of `x`. In other words, `x[[2]]` is a copy of `s`, but is *not* a slice containing `s` or its copy.

```
> x[[2]]  
[1] "aa" "bb" "cc" "dd" "ee"
```

- We can modify its content directly.

```
> x[[2]][1] = "ta"  
> x[[2]]  
[1] "ta" "bb" "cc" "dd" "ee"  
> s  
[1] "aa" "bb" "cc" "dd" "ee" # s is unaffected
```

# Variables: Data Frame

- A **data frame** is used for storing data tables. It is a list of vectors of equal length. For example, the following variable `df` is a data frame containing three vectors `n`, `s`, `b`.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc")
> b = c(TRUE, FALSE, TRUE)
> df = data.frame(n, s, b)    # df is a data frame
```

- **Build-in Data Frame**

We use built-in data frames in R for our tutorials. For example, here is a built-in data frame in R, called **mtcars**.

```
> mtcars
```

|               | mpg  | cyl | disp | hp  | drat | wt   | ... |
|---------------|------|-----|------|-----|------|------|-----|
| Mazda RX4     | 21.0 | 6   | 160  | 110 | 3.90 | 2.62 | ... |
| Mazda RX4 Wag | 21.0 | 6   | 160  | 110 | 3.90 | 2.88 | ... |
| Datsun 710    | 22.8 | 4   | 108  | 93  | 3.85 | 2.32 | ... |

.....



# Input and Display Commands

|                                                       |                                                                                                 |
|-------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| <code>x &lt;- c(1,2,4,8,16 )</code>                   | <code>#create a data vector with specified elements</code>                                      |
| <code>y &lt;- c(1:10)</code>                          | <code>#create a data vector with elements 1-10</code>                                           |
| <code>n &lt;- 10</code>                               | <code>#create a n item vector of random normal deviates</code>                                  |
| <code>x1 &lt;- c(rnorm(n))</code>                     | <code>#create another n item vector that has n added to each random uniform distribution</code> |
| <code>y1 &lt;- c(runif(n))+n</code>                   | <code>#create n samples of size "size" with probability prob from the binomial</code>           |
| <code>z &lt;- rbinom(n,size,prob)</code>              | <code>#combine them into one vector of length 2n</code>                                         |
| <code>vect &lt;- c(x,y)</code>                        | <code>#combine them into a n x 2 matrix</code>                                                  |
| <code>mat &lt;- cbind(x,y)</code>                     | <code>#display the 4th row and the 2nd column</code>                                            |
| <code>mat[4,2]</code>                                 | <code>#display the 3rd row</code>                                                               |
| <code>mat[3,]</code>                                  | <code>#display the 2nd column</code>                                                            |
| <code>mat[,2]</code>                                  | <code>#those objects meeting a logical criterion</code>                                         |
| <code>subset(dataset,logical)</code>                  | <code>#get those objects from a data frame that meet a criterion</code>                         |
| <code>subset(data.df,select=variables,logical)</code> | <code>#yet another way to get a subset</code>                                                   |
| <code>data.df[data.df=logical]</code>                 | <code>#sort a dataframe by the order of the elements in B</code>                                |
| <code>x[order(x\$B),]</code>                          | <code>#sort the dataframe in reverse order</code>                                               |
| <code>x[rev(order(x\$B)),]</code>                     |                                                                                                 |

# Moving Around Commands

```
ls()          #list the variables in the workspace
rm(x)         #remove x from the workspace
rm(list=ls()) #remove all the variables from the workspace
attach(mat)   #make the names of the variables in the matrix or data frame available in the workspace
detach(mat)   #releases the names (remember to do this each time you attach something)
with(mat, ....) #a preferred alternative to attach ... detach
new <- old[,-n] #drop the nth column
new <- old[-n,] #drop the nth row
new <- old[,-c(i,j)] #drop the ith and jth column
new <- subset(old,logical) #select those cases that meet the logical condition
complete <- subset(data.df,complete.cases(data.df))
              #find those cases with no missing values
new <- old[n1:n2,n3:n4] #select the n1 through n2 rows of variables n3 through n4)
```

# Arithmetic Operators

| Operator       | Description                 |
|----------------|-----------------------------|
| +              | addition                    |
| -              | subtraction                 |
| *              | multiplication              |
| /              | division                    |
| <b>^ or **</b> | exponentiation              |
| <b>x %% y</b>  | modulus (x mod y) 5%%2 is 1 |
| <b>x %/% y</b> | integer division 5%/2 is 2  |

2^10

[1] 1024

2\*\*10

[1] 1024

# Logical Operators

| Operator  | Description              |
|-----------|--------------------------|
| <         | less than                |
| <=        | less than or equal to    |
| >         | greater than             |
| >=        | greater than or equal to |
| ==        | exactly equal to         |
| !=        | not equal to             |
| !x        | Not x                    |
| x   y     | x OR y                   |
| x & y     | x AND y                  |
| isTRUE(x) | test if X is TRUE        |

```
x <- c(1:10)
x
1 2 3 4 5 6 7 8 9 10
x > 8
F F F F F F F T T
x < 5
T T T T F F F F F
```

# Built-in Numeric Functions

| Function                      | Description                           |
|-------------------------------|---------------------------------------|
| <b>abs(x)</b>                 | absolute value                        |
| <b>sqrt(x)</b>                | square root                           |
| <b>ceiling(x)</b>             | ceiling(3.475) is 4                   |
| <b>floor(x)</b>               | floor(3.475) is 3                     |
| <b>trunc(x)</b>               | trunc(5.99) is 5                      |
| <b>round(x, digits=n)</b>     | round(3.475, digits=2) is 3.48        |
| <b>signif(x, digits=n)</b>    | signif(3.475, digits=2) is 3.5        |
| <b>cos(x), sin(x), tan(x)</b> | also acos(x), cosh(x), acosh(x), etc. |
| <b>log(x)</b>                 | natural logarithm                     |
| <b>log10(x)</b>               | common logarithm                      |
| <b>exp(x)</b>                 | $e^x$                                 |



# Built-in Statistical Functions

## Function

**mean**(x, trim=0,  
na.rm=FALSE)

## Description

mean of object x  
# trimmed mean, removing any missing values and  
# 5 percent of highest and lowest scores  
mx <- mean(x,trim=.05,na.rm=TRUE)

**sd**(x)

standard deviation of object(x). also look at var(x)  
for variance and mad(x) for median absolute  
deviation.

**median**(x)

median

**quantile**(x, probs)

quantiles where x is the numeric vector whose  
quantiles are desired and probs is a numeric vector  
with probabilities in [0,1].  
# 30th and 84th percentiles of x  
y <- quantile(x, c(.3,.84))

**range**(x)

range

**sum**(x)

sum

**diff**(x, lag=1)

lagged differences, with lag indicating which lag to  
use

**min**(x)

minimum

**max**(x)

maximum

**scale**(x, center=TRUE, scale=TRUE)

column center or standardize a matrix

# Built-in Statistical Probability Functions

| Function                                                                                                                         | Description                                                                                                                                                                                                                                                     |
|----------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>dnorm(x)</b>                                                                                                                  | normal density function (by default m=0 sd=1)<br># plot standard normal curve<br>x <- pretty(c(-3,3), 30)<br>y <- dnorm(x)<br>plot(x, y, type='l', xlab="Normal Deviate", ylab="Density", yaxs="i")                                                             |
| <b>pnorm(q)</b>                                                                                                                  | cumulative normal probability for q<br>(area under the normal curve to the left of q)<br>pnorm(1.96) is 0.975                                                                                                                                                   |
| <b>qnorm(p)</b>                                                                                                                  | normal quantile.<br>value at the p percentile of normal distribution<br>qnorm(.9) is 1.28 # 90th percentile                                                                                                                                                     |
| <b>rnorm(n, m=0,sd=1)</b>                                                                                                        | n random normal deviates with mean m<br>and standard deviation sd.<br>#50 random normal variates with mean=50, sd=10<br>x <- rnorm(50, m=50, sd=10)                                                                                                             |
| <b>dbinom(x, size, prob)</b><br><b>pbinom(q, size, prob)</b><br><b>qbinom(p, size, prob)</b><br><b>rbinom(n, size, prob)</b>     | binomial distribution where size is the sample size<br>and prob is the probability of a heads (pi)<br># prob of 0 to 5 heads of fair coin out of 10 flips<br>dbinom(0:5, 10, .5)<br># prob of 5 or less heads of fair coin out of 10 flips<br>pbinom(5, 10, .5) |
| <b>dpois(x, lamda)</b><br><b>ppois(q, lamda)</b><br><b>qpois(p, lamda)</b><br><b>rpois(n, lamda)</b>                             | poisson distribution with m=std=lamda<br>#probability of 0,1, or 2 events with lamda=4<br>dpois(0:2, 4)<br># probability of at least 3 events with lamda=4<br>1- ppois(2,4)                                                                                     |
| <b>dunif(x, min=0, max=1)</b><br><b>punif(q, min=0, max=1)</b><br><b>qunif(p, min=0, max=1)</b><br><b>runif(n, min=0, max=1)</b> | uniform distribution, follows the same pattern<br>as the normal distribution above.<br>#10 uniform random variates<br>x <- runif(10)                                                                                                                            |

# Built-in Character Functions

| Function                                                             | Description                                                                                                                                                                                                                                            |
|----------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>substr(x, start=n1, stop=n2)</b>                                  | Extract or replace substrings in a character vector.<br>x <- "abcdef"<br>substr(x, 2, 4) is "bcd"<br>substr(x, 2, 4) <- "22222" is "a222ef"                                                                                                            |
| <b>grep(pattern, x, ignore.case=FALSE, fixed=FALSE)</b>              | Search for <i>pattern</i> in <i>x</i> . If fixed =FALSE then <i>pattern</i> is a <a href="#">regular expression</a> . If fixed=TRUE then <i>pattern</i> is a text string. Returns matching indices.<br>grep("A", c("b","A","c"), fixed=TRUE) returns 2 |
| <b>sub(pattern, replacement, x, ignore.case =FALSE, fixed=FALSE)</b> | Find <i>pattern</i> in <i>x</i> and replace with <i>replacement</i> text. If fixed=FALSE then <i>pattern</i> is a regular expression.<br>If fixed = T then <i>pattern</i> is a text string.<br>sub("\\s",".", "Hello There") returns "Hello.There"     |
| <b>strsplit(x, split)</b>                                            | Split the elements of character vector <i>x</i> at <i>split</i> .<br>strsplit("abc", "") returns 3 element vector "a","b","c"                                                                                                                          |
| <b>paste(..., sep="")</b>                                            | Concatenate strings after using <i>sep</i> string to separate them.<br>paste("x",1:3,sep="") returns c("x1","x2" "x3")<br>paste("x",1:3,sep="M") returns c("xM1","xM2" "xM3")<br>paste("Today is", date())                                             |
| <b>toupper(x)</b>                                                    | Uppercase                                                                                                                                                                                                                                              |
| <b>tolower(x)</b>                                                    | Lowercase                                                                                                                                                                                                                                              |

# Some Useful Built-in Functions

## Function

**seq**(*from* , *to*, *by*)

**rep**(*x*, *ntimes*)

**cut**(*x*, *n*)

## Description

generate a sequence

```
indices <- seq(1,10,2)
```

#indices is c(1, 3, 5, 7, 9)

repeat *x* *n* times

```
y <- rep(1:3, 2)
```

# y is c(1, 2, 3, 1, 2, 3)

divide continuous variable in factor  
with *n* levels

```
y <- cut(x, 5)
```

# Import Data : .csv and .xls(x)

The screenshot shows the RStudio interface with the 'Environment' tab selected. The 'Import Dataset' menu is open, showing options: 'From CSV...', 'From Excel...', 'From SPSS...', 'From SAS...', and 'From Stata...'. A blue callout bubble points to the 'Environment' tab with the text 'Click on Environment tab'. Another blue callout bubble points to the 'Import Dataset' menu with the text 'Select data format'. The bottom pane shows a file explorer view of the directory '~/.Data Science Training fall'17/Learning R/'.

Click on Environment tab

Select data format

| Name                                           | Size     | Modified               |
|------------------------------------------------|----------|------------------------|
| ..                                             |          |                        |
| Chat With the GSA IT Service Desk.html         | 155.8 KB | Oct 12, 2017, 1:54 PM  |
| Chat With the GSA IT Service Desk_files        |          |                        |
| Chris Olsen Intro to Statistics with Excel.pdf | 7.1 MB   | Sep 7, 2017, 10:41 AM  |
| Data Visualization.pptx                        | 3.5 MB   | Sep 13, 2017, 5:14 PM  |
| Intro to Statistics.pptx                       | 1.1 MB   | Sep 6, 2017, 10:12 AM  |
| Iris for R testing.csv                         | 4.3 KB   | Oct 11, 2017, 4:07 PM  |
| Iris for R testing.xlsx                        | 13.3 KB  | Sep 18, 2017, 1:04 PM  |
| Learning R                                     |          |                        |
| RStudio in VDI Implementation.pptx             | 305 KB   | Sep 14, 2017, 3:34 PM  |
| RStudio in VDI tutorial.pptx                   | 297.7 KB | Sep 12, 2017, 4:10 PM  |
| Stat Models in R.pdf                           | 737.5 KB | Oct 12, 2017, 11:20 AM |
| Statistics Exercises.xlsx                      | 57.6 KB  | Sep 8, 2017, 10:40 AM  |
| TufteCoversheet.pdf                            | 4.9 MB   | Sep 1, 2017, 3:03 PM   |



# Wrangle Data

The screenshot shows the RStudio 'Import Text Data' dialog box. The 'File/Url' field is set to '~\Data Science Training fall'17\Iris for R testing.csv'. The 'Data Preview' table shows the first 13 rows of the Iris dataset. The 'Import Options' section is expanded, showing settings for the dataset name, skip rows, and various parsing options. The 'Code Preview' section shows the R code generated for the import.

**Browse to File**

**Change Data Types**

**Import Options**

| Index<br>(integer) | Sepal.Length<br>(double) | Sepal.Width<br>(double) | Petal.Length<br>(double) | Petal.Width<br>(double) | Species<br>(character) |
|--------------------|--------------------------|-------------------------|--------------------------|-------------------------|------------------------|
| 1                  | 5.1                      | 3.5                     |                          |                         |                        |
| 2                  | 4.9                      | 3.0                     | 1.4                      | 0.2                     | setosa                 |
| 3                  | 4.7                      | 3.2                     | 1.3                      | 0.2                     | setosa                 |
| 4                  | 4.6                      | 3.1                     | 1.5                      | 0.2                     | setosa                 |
| 5                  | 5.0                      | 3.6                     | 1.4                      | 0.2                     | setosa                 |
| 6                  | 5.4                      | 3.9                     | 1.7                      | 0.4                     | setosa                 |
| 7                  | 4.6                      | 3.4                     | 1.4                      | 0.3                     | setosa                 |
| 8                  | 5.0                      | 3.4                     | 1.5                      | 0.2                     | setosa                 |
| 9                  | 4.4                      | 2.9                     | 1.4                      | 0.2                     | setosa                 |
| 10                 | 4.9                      | 3.1                     | 1.5                      | 0.1                     | setosa                 |
| 11                 | 5.4                      | 3.7                     | 1.5                      | 0.2                     | setosa                 |
| 12                 | 4.8                      | 3.4                     | 1.6                      | 0.2                     | setosa                 |
| 13                 | 4.8                      | 3.0                     | 1.4                      | 0.1                     | setosa                 |

Import Options:

Name: dataset  
Skip: 0

☒ First Row as Names  
☒ Trim Spaces  
☒ Open Data Viewer

Delimiter: Comma  
Quotes: Default  
Local: Configure...

Escape: None  
Comment: Default  
NA: Default

Code Preview:

```
library(readr)  
dataset <- read_csv(NULL)  
view(dataset)
```

Import Cancel

# Data Viewer for Simple Exploratory Data Analysis

The screenshot displays the RStudio environment with the following components:

- Data Viewer:** Shows a table of the first 12 rows of the 'iris' dataset. A blue callout labeled "Filter" points to the filter controls above the table.
- Console:** Contains the commands `> view(iris)` and `> plot(iris$Sepal.Length)`. A blue callout labeled "View() w/ capital V - opens data viewer" points to the `view(iris)` command.
- Plots:** A scatter plot of 'iris\$Sepal.Length' versus 'Index'. A blue callout labeled "View plots" points to the plot area.
- Environment:** Shows the 'Global Environment' with the message "Environment is empty".

|    | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|----|--------------|-------------|--------------|-------------|---------|
| 1  | 4.3          | 7.9         | 1.4          | 0.2         | setosa  |
| 2  |              |             | 1.4          | 0.2         | setosa  |
| 3  | 4.7          | 3.2         |              | 0.2         | setosa  |
| 4  | 4.6          | 3.1         | 1.5          |             | setosa  |
| 5  | 5.0          | 3.6         | 1.4          | 0.2         | setosa  |
| 6  | 5.4          | 3.9         | 1.7          | 0.4         | setosa  |
| 7  | 4.6          | 3.4         | 1.4          | 0.3         | setosa  |
| 8  | 5.0          | 3.4         | 1.5          | 0.2         | setosa  |
| 9  | 4.4          | 2.9         | 1.4          | 0.2         | setosa  |
| 10 | 4.9          | 3.1         | 1.5          | 0.1         | setosa  |
| 11 | 5.4          | 3.7         | 1.5          | 0.2         | setosa  |
| 12 | 4.8          | 3.4         | 1.6          | 0.2         | setosa  |

```
> view(iris)
> plot(iris$Sepal.Length)
>
```

# Plotting Exercise

## View(iris)

- **Histogram**

`hist(iris$Petal.Length)`

- **Scatter plot**

`plot(iris$Sepal.Length, type = 'p'), OR`

`plot(iris$Sepal.Length)`

`plot(iris$Sepal.Length, iris$Sepal.Width, type = 'p')`

- **Line chart**

`plot(iris$Sepal.Length, type = 'l')`

- **Box plot**

`boxplot(iris$Sepal.Length ~ iris$Species)`

- **Bar chart**

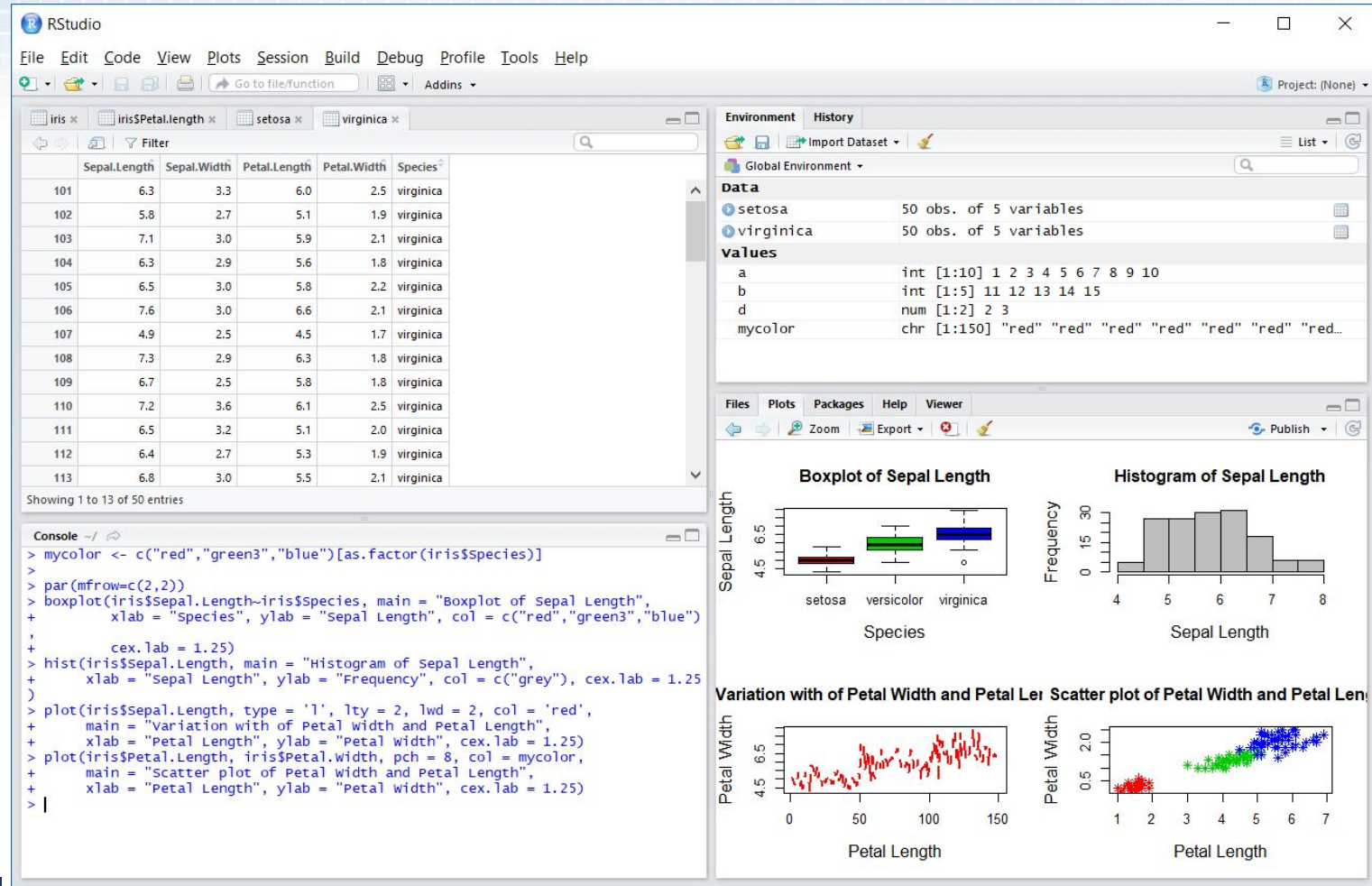
- **Customize titles and axis labels**

`plot(iris$Petal.Length, iris$Petal.Width, main = "Edgar Anderson's Iris Data", xlab = "Petal Length", ylab = "Petal Width")`

# Summary of Plotting Commands

- High level graphical commands create the plot
  - `plot( )` Scatter plot, and general plotting
  - `hist( )` Histogram
  - `stem( )` Stem-and-leaf
  - `boxplot( )` Boxplot
  - `qqnorm( )` Normal probability plot
  - `mosaicplot( )` Mosaic plot 2
- Low level graphical commands add to the plot
  - `points( )` Add points
  - `lines( )` Add lines
  - `text( )` Add text
  - • `abline( )` Add lines
  - • `legend( )` Add legend
- Most commands accept additional graphical parameters `par( )`  
Set parameters for plotting
  - `cex` Font size
  - `col` Color of plotting symbols
  - `lty` Line type
  - `lwd` Line width
  - `mar` Inner margins
  - `mfrow` Splits plotting area (mult. figs. per page) 16
  - `oma` Outer margins
  - `pch` Plotting symbol
  - `xlim` Min and max of X axis range
  - `ylim` Min and max of Y axis range

# Example of R Plotting Capabilities





# Command to Create the Example

```
> mycolor <- c("red","green3","blue")[as.factor(iris$Species)]
>
> par(mfrow=c(2,2))
> boxplot(iris$Sepal.Length~iris$Species, main = "Boxplot of Sepal Length",
+       xlab = "Species", ylab = "Sepal Length", col = c("red","green3","blue"),
+       cex.lab = 1.25)
> hist(iris$Sepal.Length, main = "Histogram of Sepal Length",
+     xlab = "Sepal Length", ylab = "Frequency", col = c("grey"), cex.lab = 1.25)
> plot(iris$Sepal.Length, type = 'l', lty = 2, lwd = 2, col = 'red',
+     main = "Variation with of Petal Width and Petal Length",
+     xlab = "Petal Length", ylab = "Petal Width", cex.lab = 1.25)
> plot(iris$Petal.Length, iris$Petal.Width, pch = 8, col = mycolor,
+     main = "Scatter plot of Petal Width and Petal Length",
+     xlab = "Petal Length", ylab = "Petal Width", cex.lab = 1.25)
```

# Extension Packages

## Open ServiceNow ticket to request a new Package on DSVD

The screenshot shows the RStudio interface with the 'Install Packages' dialog box open. The dialog box has the following fields and options:

- Install from:** Repository (CRAN, CRANextra)
- Packages (separate multiple with space or comma):** (empty text box)
- Install to Library:** C:/Users/mashk/Documents/R/win-library/3.3 [Default]
- ☒ Install dependencies
- Buttons:** Install, Cancel

The 'Packages' tab in the bottom right pane is active, showing a list of installed and available packages. A blue arrow points from the 'Install' button in the dialog box to the 'Packages' tab. Another blue arrow points from the 'Packages' tab to the 'Scroll down to review installed packages' callout.

**Callouts:**

- In your personal version only! Click Install** (points to the 'Install' button in the dialog box)
- Click on Packages tab** (points to the 'Packages' tab in the bottom right pane)
- Scroll down to review installed packages** (points to the list of packages in the 'Packages' tab)

| Name       | Description                                             | Version  |
|------------|---------------------------------------------------------|----------|
| assertthat | Easy Pre and Post Assertions                            | 0.2.0    |
| backports  | Reimplementations of Functions Introduced Since R-3.0.0 | 1.1.0    |
| base64enc  | Tools for base64 encoding                               | 0.1-3    |
| BH         | Boost C++ Header Files                                  | 1.62.0-1 |
| bindr      | Parametrized Active Bindings                            | 0.1      |
| bindrcpp   | An 'Rcpp' Interface to Active Bindings                  | 0.2      |
| bitops     | Bitwise Operations                                      | 1.0-6    |
| car        | Companion to Agresti's Categorical Data Analysis        | 2.1-5    |
| caTools    | Tools: model checking, ROC, Base64, ROC AUC, etc.       | 1.17.1   |
| cellranger | Tools: manipulate cell ranges to rows and columns       | 1.1.0    |
| colorspace | Tools: color manipulation                               | 1.3-2    |
| cvTools    | Tools: validation tools for regression models           | 0.3.2    |
| data.table | Extension of 'data.frame'                               | 1.10.4   |
| deoptim    | Differential Evolution Optimization in Pure R           | 1.0-8    |
| diagram    | Tools: Schemes for Dichromats                           | 2.0-0    |



# Help

The screenshot shows the RStudio interface with the following components:

- Top Menu Bar:** File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help.
- Top Toolbar:** Includes icons for saving, running, and other functions.
- Source Editor:** Contains a script titled "Intro to R for 10-2-2017.Rpres".
- Environment Panel:** Shows the "Global Environment" with a search bar.
- Help Panel:** The "Help" tab is selected, displaying the "Encode the Terminal Times of Time Series" documentation. A search bar is present at the top of this panel.
- Console:** Located at the bottom left, showing the command prompt.

Three callout boxes provide instructions:

- Click Help tab:** Points to the "Help" tab in the top toolbar.
- Enter search term:** Points to the search bar in the Help panel.
- Or type in the console:** Points to the console window, containing the following text:

```
> help() #opens help in the  
right bottom window  
> help("if") #opens help on  
the subject "if"
```

# Useful Links

- <https://www.r-project.org/>
- <https://www.rstudio.com/products/RStudio/>
- <http://www.statmethods.net/r-tutorial/index.html>
- [R commands](#)
  - <https://www.personality-project.org/r/r.commands.html>
- [Plotting in R](#)
  - [http://gfc.ucdavis.edu/events/arusha2016/\\_static/labs/day2/day2\\_ab2a\\_graphics.pdf](http://gfc.ucdavis.edu/events/arusha2016/_static/labs/day2/day2_ab2a_graphics.pdf)
- <http://www.cyclismo.org/tutorial/R/plotting.html>

# Q & A