**SIE 512**

**R and R Markdown Basics**

An R Markdown file allows you to combine narrative text with code. Markdown is a formatting syntax for authoring HTML, PDF or word documents. It has a notebook interface that makes it easy to configure text chunks combined with code chunks. In an R markdown file (extension .Rmd) you can add text (with markdown formatting) and embed R code chunks. When you render your .Rmd file, R Markdown will run each code chunk and embed the results beneath the code chunk. The .Rmd file can be rendered with a package called knitr which executes all of the R code chunks and markdown. In RStudio, knitr is automated for you through the “Knit” button (Ctrl+Shift+K) which will render the document and display a preview of it.

**Installing packages in RStudio:**

To add a package in RStudio: Select the packages tab, type the name of the package in the packages field and it will download and install automatically. You can also do this programmatically using the function “install. packages”

To install the R Markdown package from CRAN in Rstudio use the following:

install.packages("rmarkdown")

**Open a new markdown file**

To open a new .Rmd file in the RStudio go to File > New File > R Markdown…

An example Rmarkdown file opens.

There are three parts to an .Rmd file:

* **Header:** The text at the top of the document, written in *YAML* format.
* **Markdown sections:** Text that describes your workflow written using *markdown syntax*.
* **Code chunks:** Chunks of R code that can be run and also can be rendered using knitr to an output document.

### YAML Header

An R Markdown file always starts with a header written using [YAML syntax](about:blank). This header is sometimes referred to as the front matter.

There are four default elements in the RStudio YAML header:

* **title:** The title of your document. Note, this is not the same as the file name.
* **author:** Who wrote the document.
* **date:** By default this is the date that the file is created.
* **output:** What format will the output be in.

A YAML header begins and ends with three dashes. Also notice that the value for each element, title, author, etc, is in quotes "value-here" next to the element

### Some markdown basics:

Markdown is plain text that is styled using special characters.

Using hash signs creates headers of different levels. A single hash indicates the top-level header, two hash signs indicate a second level header and so on.

To bold a heading or \*\*word\*\* encase it with two “\*”

To italicize a \*word\* encase it in single “\*”

#### Create lists:

Use items preceded with a dash:

- step1

- step2

- step3

### Embed equations in LaTex or MathML:

$\frac{1}{n}$

$\sum\_{i=1}^{n} x\_{i}$

More markdown will be introduced as we go through this document. For example, the double asterisks in the next line will render the enclosed text in bold in markdown.

### Adding code chunks

An R code chunk is indicated by enclosing it with three back ticks and {r}

```{r}

```

The easiest way to add an R code chunk in RStudio is to use the green Insert button. It is good practice to give chunks names. e.g.

```{r load\_packages, include=FALSE}

library(ggplot2)

```

The output of code chunks can be customized with options. These options allow you to customize how or if you want code to be processed or appear on the rendered output. These are set in the {} of a chunk header. There are five common arguments:

* include = FALSE prevents code and results from appearing in the finished file. R Markdown still runs the code in the chunk, and the results can be used by other chunks.
* echo = FALSE prevents code, but not the results from appearing in the finished file. This is a useful way to embed figures.
* message = FALSE prevents messages that are generated by code from appearing in the finished file.
* warning = FALSE prevents warnings that are generated by code from appearing in the finished.
* fig.width = 4 sets figure width
* fig.height = 4 sets figure height

You can set global chunk options which become the defaults for the rest of the document. Then if you want a particular chunk to have a different behavior set these in the individual chunks. The example belwo is the default global settings in the example. rmd.

```{r setup, include=FALSE }

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

```

Another example for global options:

```{r global-options, include=FALSE}

knitr::opts\_chunk$set(fig.width=4, fig.height=4,

echo=FALSE, warning=FALSE, message=FALSE)

```

R Code

You can also evaluate R expressions inline by enclosing the expression within a single back-tick qualified with ‘r’. For example, the following code embeds R results as text in the output. e.g.

the mean is: `r mean(c(2,4,10))`

**Getting help:**

Use the help function to get information on the function.

help("mean")

### Variables and assignment:

R can work like a calculator so type 4+5 and it responds with the result. More often you will want to assign results to variables.

Variables are any names or symbols you use to assign values to. There are two common ways of assigning objects to values in R : with <- and =. In most (but not all) contexts, they can be used interchangeably. Regardless of which operator you prefer, consistency is key. I use the = operator for assignment, as it’s faster to type and more consistent with other languages.

Create a code block and add the following statements:

a=3 #assign the value 3 to a – note use of hash sign to add a comment in a code block

b=4

name = "foo" # note character values need to be in quotes

Values can be numeric, character, logical (true, false), or more complex objects like data frames. In R everything is an object of a certain class. All basic values are stored as a vector, a one-dimensional array of n values of a certain type. So a single number is a vector (of length 1).

You can check variable for class type using the function class

Testing variables for type - returns a logical variable type

Create a code block and add the following statements:

class(name)

is.numeric(a)

is.numeric(name)

Once you have assigned values to variables you can perform operations on the variables.

a+b, a\*b, a^b, b^a # the hat sign indicates to the power

or assign the result of an operation to a new variable

c<-a^b

Creating vectors (n>1)- assign a sequence of values to a variable. The c function refers to concatenation. Below, the set of numbers are concatenated to a vector d

d<-c(2.5, 3.7, 3.5, 2.9, 2.7, 3.1, 2.8, 2.4, 1.9)

You can perform operations on vectors.

mean(d), max(d), min(d), sum(d)

Get the length of the vector

length(d)

You can refer to individual elements of a vector using an index

d[2] # returns the second element in the vector

d[-3] #returns all except the third element

d[4:5] #returns the 4th and 5th elements

You can apply a function to a subset of a vector using an index

sum(d), sum(d[-3])

You can test for conditions on vectors, e.g elements > 3

test2 = (d >3)

Using conditions can be used to subset data

d[test2]

You can use the sort function to order elements of a vector.

sort(d)

Creating vectors as sequences. You can use the colon with start and end values or the seq function. The seq function has parameters from, to, by and length

f=(1:10)

f=seq(1,10), f=seq(1,10, length=50), f=seq(1,10, by=2)

###Working with Matrices

To create a matrix, use the matrix() function. The function has parameters data, nrow, ncol.

matrix(1:9), m= matrix(1:9, ncol=3)

Check the dimension of the matrix with dim

dim(m)

g=(1:21)

m=matrix (g, nrow=3)

###Operations on matrices:

rowSums, colSums, colMeans,

apply(m,1,sum) #same as rowSums apply(m,2,sum) # same as colSums

h=c(3,3,3)

J=diag(h) #Creates a diagonal matrix with elements of h in the principal diagonal

I= diag(3) # If the parameter is a scalar (k), this creates a k x k identity matrix

solve (J) #produces the inverse of J where J is a square matrix)

###Factors:

The factor function encodes a vector as nominal (categorical) variables with a set of known possible values called levels. They can be created using the factor or as.factor functions. In R you typically need to convert (cast) a character variable to a factor to identify groups for use in statistical tests and models.

fish.type= factor(c('Cod', 'Haddock', 'Flounder', 'Cod', 'Cod', 'Cod'),levels=c('Cod','Haddock', 'Flounder'))

summary (fish.type) #tabulates numbers of each level

table(fish.type) # also tabulates numbers of each level

Lists

The list function creates a list of values - that can be of different data types

entry = list(name='Jane', age='20', major='Biology')

Missing Values: All basic data types can have "missing values". These are represented by the symbol NA for "not available". For example, we can have vector 'k'

k <- c(4, NA, 5, 8, NA, 2)

Some functions will not work when NA values are present. Try mean(k).

To get around that you can add the following: na.rm = TRUE

Data Frames:

A data frame is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.).

p <- c(1,2,3,4, 5)

q <- c("green", "red", "green", NA, "red")

r <- c(TRUE,TRUE,TRUE,FALSE, TRUE)

mydata <- data.frame(p,q,r) # creates a dataframe from the three vectors

summary(mydata) #provides a summary of each column in the data frame

names(mydata) <- c("ID","Color","Passed") # assign variable names to columns

str(mydata) #displays the internal structure of an R object

\*\* Loading packages\*\*

Once you have added packages in RStudio you will always need to load them to use them. You can do this step programmatically using the library function. e.g library(ggplot2)

Install the package: ggplot2

Then load the package

library(ggplot2)

Plotting with ggplot2:

Hadley Wickham built ggplot2 based on a set of principles outlined in his layered grammar of graphics (inspired by Wilkinson's original grammar of graphics). The basic idea is that a statistical graphic is a mapping from data to aesthetic attributes (such as colour, shape, and size) of geometric objects (such as points, lines, and bars).

We start by telling ggplot which data (it must be a data frame) to use and how variables in the dataset will be used (e.g. as an x or y coordinate, as a coloring variable or a size variable, etc).

Scatter plots: first create some random data to plot

x=rnorm(100) # creates 100 standard random normal values

y=rnorm(100)

mydata2 <- data.frame(x,y) # make these into a dataframe

names(mydata2) <- c("X","Y") # add column names to the dataframe

Once you have a data frame you can call ggplot to plot any data in the data frame. aes - stands for aesthetics. It makes an assignment to variable in the dataframe. geom\_point specifies to plot x and y as point values.

ggplot(mydata2, aes(x=X, y=Y)) + geom\_point()

You can easily change and add components to a plot using the + sign. Below we first assign the plot to a variable g1. Then we can add a title and axes labels to g1.

Change the dot color and add title and axis labels

g1=ggplot(mydata2, aes(x=X, y=Y)) + geom\_point(col="red")

g1 + ggtitle("Randon points") + xlab("X Coordinates") + ylab("Y\_Coordinates")

We can change the transparency of all points by using the alpha argument (1 being the default of no transparency), Use size to make the points larger.

ggplot(mydata2, aes(x = X, y = Y)) +

geom\_point(alpha = 0.5, col = "red", size = 2)

You can plot lines with geom\_line.

Create some sequences

x2=seq(0,2\*pi, len=100)

y2=sin(x2)

mydata3 <- data.frame(x2,y2)# make these into a dataframe

names(mydata3) <- c("X2","Y2")

g4=ggplot(data=mydata3, aes(x=X2, y=Y2)) +

geom\_path(col="dark green", size=2)

Change the distribution of points

y2r=y2+rnorm(100,0,0.1)

y3r=y2+rnorm(100,0,0.2)

mydata4 <- cbind(mydata3,y2r, y3r)# make these into a dataframe

names(mydata4) <- c("X2","Y2","Y2R")

g2=ggplot(data=mydata4, aes(x=X2, y=Y2)) +

geom\_path(col="dark green", size=2)

g2+geom\_point(aes(x=X2, y=Y2R), col="purple", size=2)

Barcharts: use geom\_bar or geom\_col

x=c("oak", "maple", "ash") # create some data

y=c(12, 33, 6)

df <- data.frame(tree=x,count=y) # make a data frame

ggplot(data=df, aes(x=tree, y=count)) +

geom\_bar(stat="identity”)

Using geom-col creates the same plot

ggplot(data=df, aes(x=tree, y=count)) +

geom\_col()

## change size and color

ggplot(data=df, aes(x=tree, y=count)) +

geom\_bar(stat="identity", fill="darkgreen",width=.5)

## color fill according to variable

ggplot(data=df, aes(x=tree, y=count, fill=tree)) +

geom\_bar(stat="identity")

## add labels above bars

ggplot(data=df, aes(x=tree, y=count)) +

geom\_bar(stat="identity", fill="lightblue")+

geom\_text(aes(label=count), vjust=-0.3, size=2.5)

## Color in R

A color can be specified either by name (e.g.: “red”) or by hexadecimal code (e.g. : “#FF1234”). The different color systems available in R are described at this link : [colors in R](about:blank).

## use custom colors with black outline

p<-ggplot(df, aes(x=tree, y=count, fill=tree)) +

geom\_bar(stat="identity", color="black")

p+scale\_fill\_manual(values=c("#339900", "#E69F00", "#FF9933"))

## make some more data using rep function and rnorm

oak=rep("oak", 10)

maple=rep("maple", 12)

ash=rep("ash",6)

height=rnorm(28, 15, 5) # create a vector of heights

trees=c(oak,maple, ash)# combine tree vectors

treedf=data.frame(trees, height) # make a data frame

treedf

## Create a boxplot

ggplot(treedf, aes(x=trees, y=height, fill=trees)) +geom\_boxplot()

### Change colors

bp=ggplot(treedf, aes(x=trees, y=height, fill=trees)) +geom\_boxplot()

bp + scale\_fill\_manual(values=c("#339900", "#E69F00", "#FF9933"))

## add the mean with the stat\_summary function

bp=ggplot(treedf, aes(x=trees, y=height, fill=trees)) +geom\_boxplot() +stat\_summary(fun=mean, geom="point", size=2, color="blue", fill="red")

bp + scale\_fill\_manual(values=c("#339900", "#E69F00", "#FF9933"))

## Histogram plots

Histograms are created using geom\_histogram. Only an x variable is required as y is counts by default.

set.seed(1234)

x <- rnorm(200)

df2=data.frame(x)

# Histogram

hp<-ggplot(df2, aes(x=x))+ geom\_histogram()

hp

Change the outline and fill

ggplot(df2, aes(x=x))+ geom\_histogram(col='blue',fill='lightblue',bins = 20)

## Histogram add normal density function

Alternatively the y axis can be changed to density. To do this, add an aes parameter inside the geom\_histogram function that sets y to density. It can be changed back to count with aes(y =..count..).

The plot information can be assigned to a variable and additional plot items then added to this variable. Below we assign the plot to the variable g1 and then add a title to g1. This plot also demonstrates adding a normal curve (using stat\_function and the mean and standard deviation of salinity) to the plot. The histogram can be modified with different colors using fill, and col, by changing the number of bins as shown below.

g1<-ggplot(df2, aes(x)) + geom\_histogram(aes(y = ..density..), col='black',fill='orange', bins=20) + stat\_function(fun = dnorm, colour = "red",args = list(mean = mean(x),sd = sd(x)))

g1