Introduction to R

“How do you turn this thing on?”

Samuel Robinson, Ph.D.

Sep. 4 2023

Motivation

- “Why do I need to learn R?”

Motivation

- “Why do I need to learn R?”
  - Free, powerful, and very common

Motivation

- “Why do I need to learn R?”
  - Free, powerful, and very common
  - Interfaces with other languages (e.g. C++), and can help you learn other languages

Motivation

- "Why do I need to learn R?"
  - Free, powerful, and very common
  - Interfaces with other languages (e.g. C++), and can help you learn other languages
- "What is R good at?"

Motivation

- “Why do I need to learn R?”
  - Free, powerful, and very common
  - Interfaces with other languages (e.g. C++), and can help you learn other languages
- “What is R good at?”
  - Displaying data, running models, and processing data

Motivation

- "Why do I need to learn R?"
  - Free, powerful, and very common
  - Interfaces with other languages (e.g. C++), and can help you learn other languages
- "What is R good at?"
  - Displaying data, running models, and processing data
  - Writing presentations and papers

Motivation

- "Why do I need to learn R?"
  - Free, powerful, and very common
  - Interfaces with other languages (e.g. C++), and can help you learn other languages
- "What is R good at?"
  - Displaying data, running models, and processing data
  - Writing presentations and papers
  - *Keeping a record of what you've done*

Motivation

- "Why do I need to learn R?"
  - Free, powerful, and very common
  - Interfaces with other languages (e.g. C++), and can help you learn other languages
- "What is R good at?"
  - Displaying data, running models, and processing data
  - Writing presentations and papers
  - *Keeping a record of what you've done*
- "What is R bad at?"

Motivation

- “Why do I need to learn R?”
  - Free, powerful, and very common
  - Interfaces with other languages (e.g. C++), and can help you learn other languages
- “What is R good at?”
  - Displaying data, running models, and processing data
  - Writing presentations and papers
  - *Keeping a record of what you’ve done*
- “What is R bad at?”
  - No point-and-click interface; simple things can take more time

Motivation

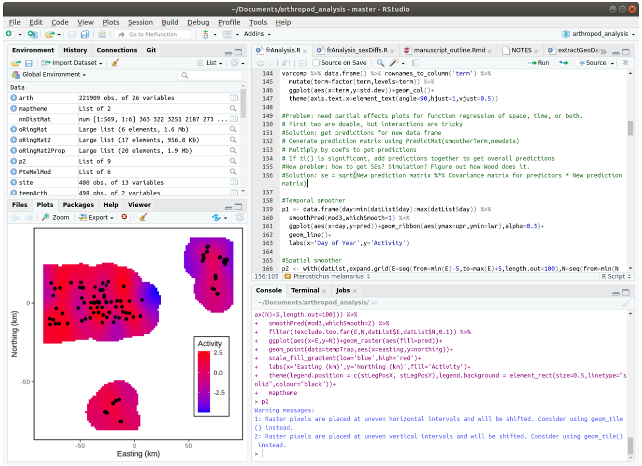
- "Why do I need to learn R?"
  - Free, powerful, and very common
  - Interfaces with other languages (e.g. C++), and can help you learn other languages
- "What is R good at?"
  - Displaying data, running models, and processing data
  - Writing presentations and papers
  - *Keeping a record of what you've done*
- "What is R bad at?"
  - No point-and-click interface; simple things can take more time
  - Can be slow if datasets are large

Motivation

- "Why do I need to learn R?"
  - Free, powerful, and very common
  - Interfaces with other languages (e.g. C++), and can help you learn other languages
- "What is R good at?"
  - Displaying data, running models, and processing data
  - Writing presentations and papers
  - *Keeping a record of what you've done*
- "What is R bad at?"
  - No point-and-click interface; simple things can take more time
  - Can be slow if datasets are large
- This is not a course in R programming (see here, here, here , but you do need some basic techniques

RStudio GUI

- The **Console** is the main input into R (where you tell it to do things)


~/Documents/arthropod_analysis - master - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
arthropod_analysis
Environment History Connections Git
Import Dataset
List
Global Environment
Files Plots Packages Help Viewer
Zoom Export
Northing (km)
Easting (km)
Activity
2.5
0.0
-2.5
0
-50
50
frAnalysis.R frAnalysis_sexDiffs.R manuscript_outline.Rmd NOTES extractGeoDa
Source on Save
Run
Source
156:105
Pterostichus melanarius
R Script
Console Terminal Jobs
~/Documents/arthropod_analysis/

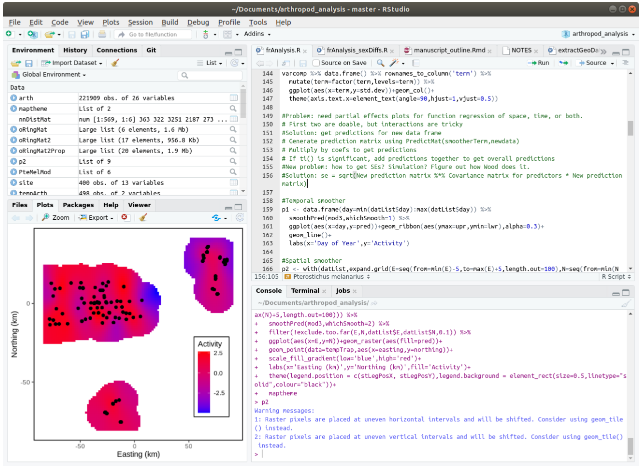
```
Data
arth           221909 obs. of 26 variables
maptheme       List of 2
nnDistMat      num [1:569, 1:6] 363 322 3251 2187 273 ...
oRingMat       Large list (6 elements, 1.6 Mb)
oRingMat2      Large list (17 elements, 956.8 Kb)
oRingMat2Prop  Large list (20 elements, 1.9 Mb)
p2             List of 9
PteMelMod      List of 6
site           400 obs. of 13 variables
tempArth       498 obs. of 2 variables
```

```
varcomp %>% data.frame() %>% rownames_to_column('term') %>%
  mutate(term=factor(term,levels=term)) %>%
  ggplot(aes(x=term,y=std.dev))+geom_col()+
  theme(axis.text.x=element_text(angle=90,hjust=1,vjust=0.5))

#Problem: need partial effects plots for function regression of space, time, or both.
# First two are doable, but interactions are tricky
#Solution: get predictions for new data frame
# Generate prediction matrix using PredictMat(smootherTerm,newdata)
# Multiply by coefs to get predictions
# If ti() is significant, add predictions together to get overall predictions
#New problem: how to get SEs? Simulation? Figure out how Wood does it.
#Solution: se = sqrt(New prediction matrix %*% Covariance matrix for predictors * New prediction
matrix)

#Temporal smoother
p1 <- data.frame(day=min(datList$day):max(datList$day)) %>%
  smoothPred(mod3,whichSmooth=1) %>%
  ggplot(aes(x=day,y=pred))+geom_ribbon(aes(ymax=upr,ymin=lwr),alpha=0.3)+
  geom_line()+
  labs(x='Day of Year',y='Activity')

#Spatial smoother
p2 <- with(datList,expand.grid(E=seq(from=min(E)-5,to=max(E)+5,length.out=100),N=seq(from=min(N
```

```
ax(N)+5,length.out=100))) %>%
+   smoothPred(mod3,whichSmooth=2) %>%
+   filter(!exclude.too.far(E,N,datList$E,datList$N,0.1)) %>%
+   ggplot(aes(x=E,y=N))+geom_raster(aes(fill=pred))+
+   geom_point(data=tempTrap,aes(x=easting,y=northing))+
+   scale_fill_gradient(low='blue',high='red')+
+   labs(x='Easting (km)',y='Northing (km)',fill='Activity')+
+   theme(legend.position = c(stLegPosX, stLegPosY),legend.background = element_rect(size=0.5,linetype="s
olid",colour="black"))+
+   maptheme
> p2
Warning messages:
1: Raster pixels are placed at uneven horizontal intervals and will be shifted. Consider using geom_tile
() instead.
2: Raster pixels are placed at uneven vertical intervals and will be shifted. Consider using geom_tile()
 instead.
> 
```

RStudio GUI

- The **Console** is the main input into R (where you tell it to do things)
- **Scripts** are lists of commands that get passed into the console



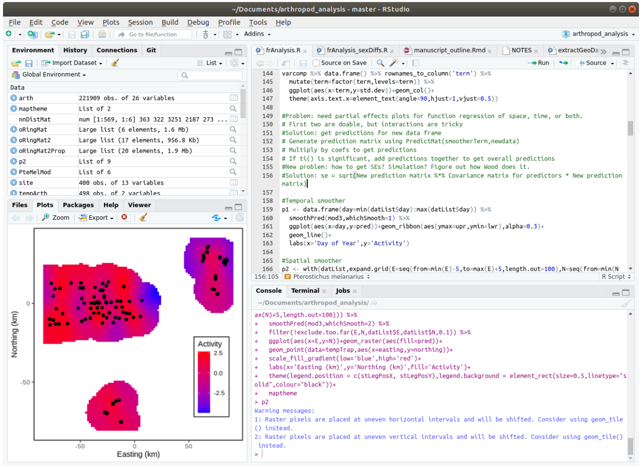
Data	
arth	221909 obs. of 26 variables
maptheme	List of 2
nnDistMat	num [1:569, 1:6] 363 322 3251 2187 273 ...
oRingMat	Large list (6 elements, 1.6 Mb)
oRingMat2	Large list (17 elements, 956.8 Kb)
oRingMat2Prop	Large list (20 elements, 1.9 Mb)
p2	List of 9
PteMelMod	List of 6
site	400 obs. of 13 variables
tempArth	498 obs. of 2 variables

```r
varcomp %>% data.frame() %>% rownames_to_column('term') %>%
  mutate(term=factor(term,levels=term)) %>%
  ggplot(aes(x=term,y=std.dev))+geom_col()+
  theme(axis.text.x=element_text(angle=90,hjust=1,vjust=0.5))

#Problem: need partial effects plots for function regression of space, time, or both.
# First two are doable, but interactions are tricky
#Solution: get predictions for new data frame
# Generate prediction matrix using PredictMat(smootherTerm,newdata)
# Multiply by coefs to get predictions
# If ti() is significant, add predictions together to get overall predictions
#New problem: how to get SEs? Simulation? Figure out how Wood does it.
#Solution: se = sqrt(New prediction matrix %*% Covariance matrix for predictors * New prediction
matrix)

#Temporal smoother
p1 <- data.frame(day=min(datList$day):max(datList$day)) %>%
  smoothPred(mod3,whichSmooth=1) %>%
  ggplot(aes(x=day,y=pred))+geom_ribbon(aes(ymax=upr,ymin=lwr),alpha=0.3)+
  geom_line()+
  labs(x='Day of Year',y='Activity')

#Spatial smoother
p2 <- with(datList,expand.grid(E=seq(from=min(E)-5,to=max(E)+5,length.out=100),N=seq(from=min(N
```

```
ax(N)+5,length.out=100))) %>%
+   smoothPred(mod3,whichSmooth=2) %>%
+   filter(!exclude.too.far(E,N,datList$E,datList$N,0.1)) %>%
+   ggplot(aes(x=E,y=N))+geom_raster(aes(fill=pred))+
+   geom_point(data=tempTrap,aes(x=easting,y=northing))+
+   scale_fill_gradient(low='blue',high='red')+
+   labs(x='Easting (km)',y='Northing (km)',fill='Activity')+
+   theme(legend.position = c(stLegPosX, stLegPosY),legend.background = element_rect(size=0.5,linetype="s
olid",colour="black"))+
+   maptheme
> p2
Warning messages:
1: Raster pixels are placed at uneven horizontal intervals and will be shifted. Consider using geom_tile
() instead.
2: Raster pixels are placed at uneven vertical intervals and will be shifted. Consider using geom_tile()
 instead.
> 
```

RStudio GUI

- The **Console** is the main input into R (where you tell it to do things)
- **Scripts** are lists of commands that get passed into the console
- If you're using RStudio, 2 of the 4 panes will be dedicated to the console and scripts



Data	
arth	221909 obs. of 26 variables
maptheme	List of 2
nnDistMat	num [1:569, 1:6] 363 322 3251 2187 273 ...
oRingMat	Large list (6 elements, 1.6 Mb)
oRingMat2	Large list (17 elements, 956.8 Kb)
oRingMat2Prop	Large list (20 elements, 1.9 Mb)
p2	List of 9
PteMelMod	List of 6
site	400 obs. of 13 variables
tempArth	498 obs. of 2 variables

```r
varcomp %>% data.frame() %>% rownames_to_column('term') %>%
  mutate(term=factor(term,levels=term)) %>%
  ggplot(aes(x=term,y=std.dev))+geom_col()+
  theme(axis.text.x=element_text(angle=90,hjust=1,vjust=0.5))

#Problem: need partial effects plots for function regression of space, time, or both.
# First two are doable, but interactions are tricky
#Solution: get predictions for new data frame
# Generate prediction matrix using PredictMat(smootherTerm,newdata)
# Multiply by coefs to get predictions
# If ti() is significant, add predictions together to get overall predictions
#New problem: how to get SEs? Simulation? Figure out how Wood does it.
#Solution: se = sqrt(New prediction matrix %*% Covariance matrix for predictors * New prediction
matrix)

#Temporal smoother
p1 <- data.frame(day=min(datList$day):max(datList$day)) %>%
  smoothPred(mod3,whichSmooth=1) %>%
  ggplot(aes(x=day,y=pred))+geom_ribbon(aes(ymax=upr,ymin=lwr),alpha=0.3)+
  geom_line()+
  labs(x='Day of Year',y='Activity')

#Spatial smoother
p2 <- with(datList,expand.grid(E=seq(from=min(E)-5,to=max(E)+5,length.out=100),N=seq(from=min(N
```

```
ax(N)+5,length.out=100))) %>%
+   smoothPred(mod3,whichSmooth=2) %>%
+   filter(!exclude.too.far(E,N,datList$E,datList$N,0.1)) %>%
+   ggplot(aes(x=E,y=N))+geom_raster(aes(fill=pred))+
+   geom_point(data=tempTrap,aes(x=easting,y=northing))+
+   scale_fill_gradient(low='blue',high='red')+
+   labs(x='Easting (km)',y='Northing (km)',fill='Activity')+
+   theme(legend.position = c(stLegPosX, stLegPosY),legend.background = element_rect(size=0.5,linetype="s
olid",colour="black"))+
+   maptheme
> p2
Warning messages:
1: Raster pixels are placed at uneven horizontal intervals and will be shifted. Consider using geom_tile
() instead.
2: Raster pixels are placed at uneven vertical intervals and will be shifted. Consider using geom_tile()
 instead.
> 
```

Objects and Functions

- Everything in R is either an **Object** or a **Function**. All must have a unique name, or else the *Steve Problem*[1] occurs.

[1]Two guys named Steve at a party: which one is which?

Objects and Functions

- Everything in R is either an **Object** or a **Function**. All must have a unique name, or else the *Steve Problem*[1] occurs.
- Some common **objects** (things stored in memory):

[1] Two guys named Steve at a party: which one is which?

Objects and Functions

- Everything in R is either an **Object** or a **Function**. All must have a unique name, or else the *Steve Problem*[1] occurs.
- Some common **objects** (things stored in memory):
  - Characters, Logicals, & Numerics

[1] Two guys named Steve at a party: which one is which?

Objects and Functions

- Everything in R is either an **Object** or a **Function**. All must have a unique name, or else the *Steve Problem*[1] occurs.
- Some common **objects** (things stored in memory):
  - Characters, Logicals, & Numerics
  - Vectors & Matrices

[1] Two guys named Steve at a party: which one is which?

Objects and Functions

- Everything in R is either an **Object** or a **Function**. All must have a unique name, or else the *Steve Problem*[1] occurs.
- Some common **objects** (things stored in memory):
  - Characters, Logicals, & Numerics
  - Vectors & Matrices
  - Dataframes & Lists

[1]Two guys named Steve at a party: which one is which?

Objects and Functions

- Everything in R is either an **Object** or a **Function**. All must have a unique name, or else the *Steve Problem*[1] occurs.
- Some common **objects** (things stored in memory):
  - Characters, Logicals, & Numerics
  - Vectors & Matrices
  - Dataframes & Lists
- Some common **functions** (things done to objects):

[1] Two guys named Steve at a party: which one is which?

Objects and Functions

- Everything in R is either an **Object** or a **Function**. All must have a unique name, or else the *Steve Problem*[1] occurs.
- Some common **objects** (things stored in memory):
  - Characters, Logicals, & Numerics
  - Vectors & Matrices
  - Dataframes & Lists
- Some common **functions** (things done to objects):
  - *mean*, *sd*, *median*, *quantile*, *c*, *paste*

[1] Two guys named Steve at a party: which one is which?

Objects and Functions

- Everything in R is either an **Object** or a **Function**. All must have a unique name, or else the *Steve Problem*[1] occurs.
- Some common **objects** (things stored in memory):
  - Characters, Logicals, & Numerics
  - Vectors & Matrices
  - Dataframes & Lists
- Some common **functions** (things done to objects):
  - *mean*, *sd*, *median*, *quantile*, *c*, *paste*
  - *plot*, *summary*

[1] Two guys named Steve at a party: which one is which?

Objects and Functions

- Everything in R is either an **Object** or a **Function**. All must have a unique name, or else the *Steve Problem*[1] occurs.
- Some common **objects** (things stored in memory):
  - Characters, Logicals, & Numerics
  - Vectors & Matrices
  - Dataframes & Lists
- Some common **functions** (things done to objects):
  - *mean*, *sd*, *median*, *quantile*, *c*, *paste*
  - *plot*, *summary*
    - these are polymorphic functions: they do different things to different types of objects

[1] Two guys named Steve at a party: which one is which?

Objects and Functions

- Everything in R is either an **Object** or a **Function**. All must have a unique name, or else the *Steve Problem*[1] occurs.
- Some common **objects** (things stored in memory):
  - Characters, Logicals, & Numerics
  - Vectors & Matrices
  - Dataframes & Lists
- Some common **functions** (things done to objects):
  - *mean*, *sd*, *median*, *quantile*, *c*, *paste*
  - *plot*, *summary*
    - these are polymorphic functions: they do different things to different types of objects
  - Control flow - *if* and *for*

[1] Two guys named Steve at a party: which one is which?

Objects

- Let's make some objects. These are all single objects:

```r
myString <- "Hello world"  #A string object
myNumeric <- 12345  #A numeric object
myLogical <- TRUE  #A logical object
```

Objects

- Let's make some objects. These are all single objects:

```
myString <- "Hello world"  #A string object
myNumeric <- 12345  #A numeric object
myLogical <- TRUE  #A logical object
```

- These are objects joined into a *vector*, joined by the function c (concatenate):

```
myCharVec <- c("I like pie", "I like cake", "I like anything you bake")
myNumVec <- c(1, 2, 3, 4, 5)
myLogVec <- c(TRUE, TRUE, FALSE, TRUE, FALSE, FALSE)
```

Objects

- Let's make some objects. These are all single objects:

```
myString <- "Hello world"  #A string object
myNumeric <- 12345  #A numeric object
myLogical <- TRUE  #A logical object
```

- These are objects joined into a *vector*, joined by the function c (concatenate):

```
myCharVec <- c("I like pie", "I like cake", "I like anything you bake")
myNumVec <- c(1, 2, 3, 4, 5)
myLogVec <- c(TRUE, TRUE, FALSE, TRUE, FALSE, FALSE)
```

- How long are each of these vectors?

```
howLong <- c(length(myCharVec), length(myNumVec), length(myLogVec))
howLong  #This executes the `print` command on `howLong`
```

```
## [1] 3 5 6
```

Vectors - “getting”

```r
myCharVec  #Here's what's inside the whole thing
```

```
## [1] "I like pie"                    "I like cake"
## [3] "I like anything you bake"
```

Vectors - "getting"

```r
myCharVec  #Here's what's inside the whole thing
```

```
## [1] "I like pie"                "I like cake"
## [3] "I like anything you bake"
```

- Single number:

  ```r
  myCharVec[1]
  ```

  ```
  ## [1] "I like pie"
  ```

Vectors - "getting"

```r
myCharVec  #Here's what's inside the whole thing
```

```
## [1] "I like pie"                 "I like cake"
## [3] "I like anything you bake"
```

- Single number:

  ```r
  myCharVec[1]
  ```

  ```
  ## [1] "I like pie"
  ```

- Vector of numbers

  ```r
  myCharVec[c(2, 3)]
  ```

  ```
  ## [1] "I like cake"                "I like anything you bake"
  ```

Vectors - “getting”

```
myCharVec  #Here's what's inside the whole thing
```

```
## [1] "I like pie"                "I like cake"
## [3] "I like anything you bake"
```

- Single number:

  ```
  myCharVec[1]
  ```

  ```
  ## [1] "I like pie"
  ```

- Vector of numbers

  ```
  myCharVec[c(2, 3)]
  ```

  ```
  ## [1] "I like cake"              "I like anything you bake"
  ```

- Logical vector

  ```
  myCharVec[c(TRUE, FALSE, TRUE)]
  ```

  ```
  ## [1] "I like pie"               "I like anything you bake"
  ```

Vectors - “setting”

Vectors are set in the same way, using the assignment operator: <- OR =

Vectors - “setting”

Vectors are set in the same way, using the assignment operator: <- OR =

- String vector

```r
myCharVec[c(2, 3)] <- c("Cats", "Dogs")
```

Vectors - “setting”

Vectors are set in the same way, using the assignment operator: <- OR =

- String vector

```
myCharVec[c(2, 3)] <- c("Cats", "Dogs")
```

- Logical vector

```
myCharVec[c(TRUE, FALSE, FALSE)] = "Parakeets"
```

Vectors - “setting”

Vectors are set in the same way, using the assignment operator: <- OR =

- String vector

```
myCharVec[c(2, 3)] <- c("Cats", "Dogs")
```

- Logical vector

```
myCharVec[c(TRUE, FALSE, FALSE)] = "Parakeets"
```

- Results:

```
myCharVec  #Here's what's inside the whole thing
```

```
## [1] "Parakeets" "Cats"      "Dogs"
```

Class conversions

Vectors (or other data) can be converted between **classes**, usually using *as.something* statements:

Class conversions

Vectors (or other data) can be converted between **classes**, usually using *as.something* statements:

- Logical to numeric

```r
as.numeric(myLogVec)
```

```
## [1] 1 1 0 1 0 0
```

Class conversions

Vectors (or other data) can be converted between **classes**, usually using *as.something* statements:

- Logical to numeric

```r
as.numeric(myLogVec)
```

```
## [1] 1 1 0 1 0 0
```

- Numeric to character

```r
as.character(myNumVec)
```

```
## [1] "1" "2" "3" "4" "5"
```

Class conversions

Vectors (or other data) can be converted between **classes**, usually using *as.something* statements:

- Logical to numeric

```
as.numeric(myLogVec)
```

```
## [1] 1 1 0 1 0 0
```

- Numeric to character

```
as.character(myNumVec)
```

```
## [1] "1" "2" "3" "4" "5"
```

- Characters to **factors**: these represent *categories* or experimental levels

```
as.factor(myCharVec)  #Default order is alphabetical
```

```
## [1] Parakeets Cats      Dogs
## Levels: Cats Dogs Parakeets
```

Dataframes

- Dataframes look similar to matrices, but can hold different data types in each column:

```
myDF <- data.frame(stringCol = myCharVec,
    numCol = myNumVec[c(1:3)],
    logCol = myLogVec[c(1:3)])
myDF
```

```
##   stringCol numCol logCol
## 1 Parakeets      1   TRUE
## 2      Cats      2   TRUE
## 3      Dogs      3  FALSE
```

Dataframes

- Dataframes look similar to matrices, but can hold different data types in each column:

```
myDF <- data.frame(stringCol = myCharVec,
    numCol = myNumVec[c(1:3)],
    logCol = myLogVec[c(1:3)])
myDF
```

```
##   stringCol numCol logCol
## 1 Parakeets      1   TRUE
## 2      Cats      2   TRUE
## 3      Dogs      3  FALSE
```

-
  ```
  summary(myDF)  #This function summarizes each column
  ```

```
##   stringCol              numCol       logCol
##  Length:3           Min.   :1.0   Mode :logical
##  Class :character   1st Qu.:1.5   FALSE:1
##  Mode  :character   Median :2.0   TRUE :2
##                     Mean   :2.0
##                     3rd Qu.:2.5
##                     Max.   :3.0
```

Accessing Dataframes

Dataframes can be accessed numerically, by their name slots (using the $ operator), or with a mixture of the two:

Accessing Dataframes

Dataframes can be accessed numerically, by their name slots (using the $ operator), or with a mixture of the two:

-
  ```r
  myDF[1, 2]
  ```

  ```
  ## [1] 1
  ```

Accessing Dataframes

Dataframes can be accessed numerically, by their name slots (using the $ operator), or with a mixture of the two:

-
  ```r
  myDF[1, 2]
  ```

  ```
  ## [1] 1
  ```

-
  ```r
  myDF$numCol  #This gets all of the column 'numCol'
  ```

  ```
  ## [1] 1 2 3
  ```

Accessing Dataframes

Dataframes can be accessed numerically, by their name slots (using the $ operator), or with a mixture of the two:

-
  ```r
  myDF[1, 2]
  ```

  ```
  ## [1] 1
  ```

-
  ```r
  myDF$numCol  #This gets all of the column 'numCol'
  ```

  ```
  ## [1] 1 2 3
  ```

-
  ```r
  myDF[1, "numCol"]
  ```

  ```
  ## [1] 1
  ```

Accessing Dataframes

Dataframes can be accessed numerically, by their name slots (using the $ operator), or with a mixture of the two:

-
  ```r
  myDF[1, 2]
  ```
  ```
  ## [1] 1
  ```

-
  ```r
  myDF$numCol  #This gets all of the column 'numCol'
  ```
  ```
  ## [1] 1 2 3
  ```

-
  ```r
  myDF[1, "numCol"]
  ```
  ```
  ## [1] 1
  ```

-
  ```r
  myDF$numCol[1]
  ```
  ```
  ## [1] 1
  ```

Manipulating dataframe

Like other objects, you can alter dataframes

- You can add columns

```
myDF$numCol2 <- myDF$numCol * 3  #Multiplies numCol by 3
```

Manipulating dataframe

Like other objects, you can alter dataframes

- You can add columns

```
myDF$numCol2 <- myDF$numCol * 3  #Multiplies numCol by 3
```

- You can also alter columns in place, or elements within columns

```
myDF$numCol <- (myDF$numCol)^2  #Square of numCol
myDF$numCol[3] <- myDF$numCol[3] - myDF$numCol[2]  #Subtracts 2nd from 3rd
```

Manipulating dataframe

Like other objects, you can alter dataframes

- You can add columns

```
myDF$numCol2 <- myDF$numCol * 3  #Multiplies numCol by 3
```

- You can also alter columns in place, or elements within columns

```
myDF$numCol <- (myDF$numCol)^2  #Square of numCol
myDF$numCol[3] <- myDF$numCol[3] - myDF$numCol[2]  #Subtracts 2nd from 3rd
```

- You can delete columns by subsetting the dataframe, or assigning the column to `NULL`

```
myDF <- myDF[, c(1, 2)]  #Selects only column 1 and 2
myDF$numCol <- NULL  #Removes numCol
```

Manipulating dataframe

Like other objects, you can alter dataframes

- You can add columns

```r
myDF$numCol2 <- myDF$numCol * 3  #Multiplies numCol by 3
```

- You can also alter columns in place, or elements within columns

```r
myDF$numCol <- (myDF$numCol)^2  #Square of numCol
myDF$numCol[3] <- myDF$numCol[3] - myDF$numCol[2]  #Subtracts 2nd from 3rd
```

- You can delete columns by subsetting the dataframe, or assigning the column to `NULL`

```r
myDF <- myDF[, c(1, 2)]  #Selects only column 1 and 2
myDF$numCol <- NULL  #Removes numCol
```

- We'll learn easier ways of doing this next week using the `tidyverse`

Reading csv files

- One common practice is to read in your own dataframe from a csv file [2]

```r
testDat <- read.csv("test_results.csv")  #Path to csv file
head(testDat)  #head shows only the first 6 rows of dataframe
```

```
##   Concentration Treatment Lab.Member Time.of.Day
## 1           2.9   Control       Will     Morning
## 2           3.2   Control       Will     Morning
## 3           3.6   Control       Will     Morning
## 4           5.6         A       Will     Morning
## 5           6.8         A       Will     Morning
## 6           7.0         A       Will     Morning
```

[2] Excel files can be read in, but can cause problems

Reading csv files

- One common practice is to read in your own dataframe from a csv file [2]

```
testDat <- read.csv("test_results.csv")  #Path to csv file
head(testDat)  #head shows only the first 6 rows of dataframe
```

```
##   Concentration Treatment Lab.Member Time.of.Day
## 1           2.9   Control       Will     Morning
## 2           3.2   Control       Will     Morning
## 3           3.6   Control       Will     Morning
## 4           5.6         A       Will     Morning
## 5           6.8         A       Will     Morning
## 6           7.0         A       Will     Morning
```

- R can't handle spaces or other special characters in the column headers (replaces them with periods). It also tries to guess the proper data type for each column

[2] Excel files can be read in, but can cause problems

Reading csv files

- One common practice is to read in your own dataframe from a csv file [2]

```
testDat <- read.csv("test_results.csv")  #Path to csv file
head(testDat)  #head shows only the first 6 rows of dataframe
```

```
##   Concentration Treatment Lab.Member Time.of.Day
## 1           2.9   Control       Will     Morning
## 2           3.2   Control       Will     Morning
## 3           3.6   Control       Will     Morning
## 4           5.6         A       Will     Morning
## 5           6.8         A       Will     Morning
## 6           7.0         A       Will     Morning
```

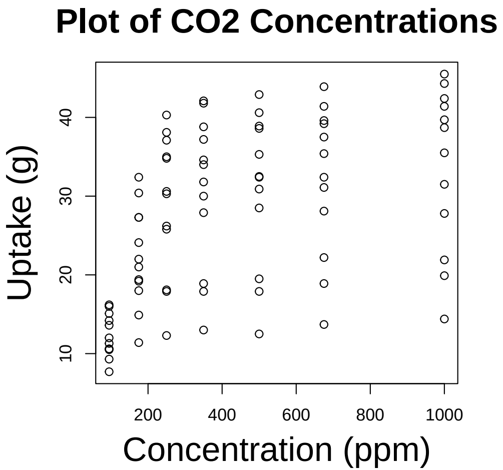
- R can't handle spaces or other special characters in the column headers (replaces them with periods). It also tries to guess the proper data type for each column
- Make sure that the csv file is in your *working directory*

[2] Excel files can be read in, but can cause problems

Plotting

- The *plot* command is useful for quickly looking at sets of data. The following CO2 dataset is built-in to R.[3]

```
#Makes a plot of the uptake (y) and
# concentration (x) columns of CO2
# dataframe, and customizes axis labels
plot(x = CO2$conc, y = CO2$uptake,
     xlab = 'Concentration (ppm)',
     ylab = 'Uptake (g)',
     main = 'Plot of CO2 Concentrations',
     cex.lab = 2, cex.main=2)
```

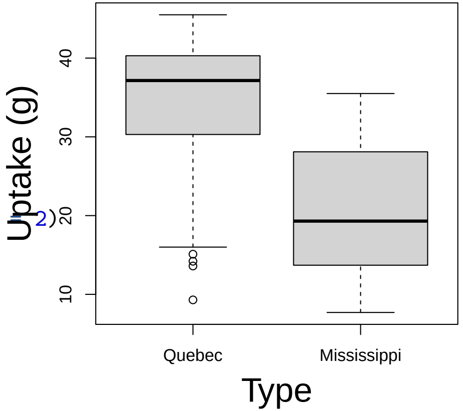


[3]To see others, type `data()` in the console

More Plotting

- The boxplot command can summarize *continuous* and *categorical* data

```r
# Boxplot uses a formula rather than
# x,y vectors

# Formula: uptake depends on (~) Type
boxplot(CO2$uptake ~ CO2$Type, xlab = "Type",
    ylab = "Uptake (g)", cex.lab = 2, cex.main = 2)
```



First challenge

Your supervisor has just given you a dataset (*test_results.csv*) recorded by two undergrads. However, these undergrads were in a hurry and have made some mistakes:

- Make a **script** in R, and use this to record what you do
- Read the *csv* file and fix any mistakes. Bonus if you do this without using Excel!
- Plot the concentration data by treatment group, then plot it for each undergrad. Does there look like much of a difference?
- Some useful commands: **read.csv**, **boxplot**, **is.na**, **as.factor**, **summary**

Functions

- Functions take objects as **arguments** (input) and return other **objects** (output)

```r
myNumVec <- c(1, 2, 3, 4, 5)
meanVec <- mean(myNumVec)  #Arithmetic mean (average)
sdVec <- sd(myNumVec)  #Standard deviation (sqrt(variance))
meanSdVec <- c(meanVec, sdVec)  #Joins mean and SD into a vector
meanSdVec
```

```
## [1] 3.000000 1.581139
```

Functions

- Functions take objects as **arguments** (input) and return other **objects** (output)

```r
myNumVec <- c(1, 2, 3, 4, 5)
meanVec <- mean(myNumVec)  #Arithmetic mean (average)
sdVec <- sd(myNumVec)  #Standard deviation (sqrt(variance))
meanSdVec <- c(meanVec, sdVec)  #Joins mean and SD into a vector
meanSdVec
```

```
## [1] 3.000000 1.581139
```

- If you can't remember how a command works, use ? in the console to access the help files (e.g. `?median`)

Homemade Functions

- You can make your own functions! This is useful if you have to do the same thing to many different input objects.

```
myFun <- function(input) {
    # Takes a vector of numbers
    A <- mean(input)  #Take the mean of INPUT
    B <- sd(input)  #Take the SD of INPUT
    C <- c(A, B)  #Join A and B into a vector C
    return(C)  #Return (output) C, then end the function
}
myFun(myNumVec)  #Same as previous slide
```

```
## [1] 3.000000 1.581139
```

Homemade Functions

- You can make your own functions! This is useful if you have to do the same thing to many different input objects.

```
myFun <- function(input) {
    # Takes a vector of numbers
    A <- mean(input)  #Take the mean of INPUT
    B <- sd(input)  #Take the SD of INPUT
    C <- c(A, B)  #Join A and B into a vector C
    return(C)  #Return (output) C, then end the function
}
myFun(myNumVec)  #Same as previous slide
```

```
## [1] 3.000000 1.581139
```

- The objects inside of functions (A, B, C in the one above) disappear after the function runs. However, *functions can see objects in the outer environment*, so beware of the Steve Problem*

Summary statistics

Often we want to get the mean of one columns, but split it up by other things in the dataframe.

Summary statistics

Often we want to get the mean of one columns, but split it up by other things in the dataframe.

- Using the CO2 plant example, how does *uptake* differ between *Type*?

```r
# Split up uptake by Type and Treatment, then take the mean
tapply(CO2$uptake, list(CO2$Type, CO2$Treatment), mean)
```

```
##             nonchilled  chilled
## Quebec        35.33333 31.75238
## Mississippi   25.95238 15.81429
```

Summary statistics

Often we want to get the mean of one columns, but split it up by other things in the dataframe.

- Using the CO2 plant example, how does *uptake* differ between *Type*?

```
# Split up uptake by Type and Treatment, then take the mean
tapply(CO2$uptake, list(CO2$Type, CO2$Treatment), mean)
```

```
##             nonchilled  chilled
## Quebec        35.33333 31.75238
## Mississippi   25.95238 15.81429
```

- Typing "CO2" over and over again is annoying. You can use *with* to avoid repeating the name of the dataframe (avoid using *attach*)

```
# Runs command inside the name space of the CO2 object
with(CO2, tapply(uptake, list(Type, Treatment), sd))
```

if statements

- R can be told to do things only *if* certain conditions apply. This is useful inside of functions for error handling:

```
myFun2 <- function(x) {
    xClass <- class(x)  #What class is x? (Numeric, character, boolean)

    if (xClass == "character") {
        # == means 'are these things equal'?
        return("This is a string")  #If x is a character, returns a message
    } else {
        return(mean(x))  #If x isn't a character, returns the mean of x
    }
}
myFun2(myCharVec)
```

```
## [1] "This is a string"
```

```
myFun2(myNumVec)
```

```
## [1] 3
```

for loops

- R can be told to do things repeatedly, using an *index* inside a loop:

```r
classVec <- rep(0, 10)  #Storage vector of zeros, 10 long
classVec[c(1, 2)] <- 1  #Set first two slots to 1

# Each time the loop repeats, i will take on values 3 to 10
for (i in 3:length(classVec)) {
    # ith slot of classVec becomes the sum of the previous two slots
    classVec[i] <- classVec[i - 1] + classVec[i - 2]
}
```

for loops

- R can be told to do things repeatedly, using an *index* inside a loop:

```
classVec <- rep(0, 10)  #Storage vector of zeros, 10 long
classVec[c(1, 2)] <- 1  #Set first two slots to 1

# Each time the loop repeats, i will take on values 3 to 10
for (i in 3:length(classVec)) {
    # ith slot of classVec becomes the sum of the previous two slots
    classVec[i] <- classVec[i - 1] + classVec[i - 2]
}
```

-
  ```
  classVec  #First 10 numbers in the Fibbonaci sequence
  ```

```
##  [1]  1  1  2  3  5  8 13 21 34 55
```

Second challenge

Population growth models are common in ecology, and usually often take the form $n_t = n_{t-1} + rn_{t-1}$, where n is the number of critters at some time point t, and r is the change in n from one point to the next (r = 0: no change).

- Write a `function` (with a `for` loop inside) that performs a simple population simulation using the following models:

Hint: functions need input variables that tell them what to do. The input variables here could be things like *starting population* (n_0), *growth rate* (r), and *number of time steps* (T) to simulate

Second challenge

Population growth models are common in ecology, and usually often take the form $n_t = n_{t-1} + rn_{t-1}$, where n is the number of critters at some time point t, and r is the change in n from one point to the next (r = 0: no change).

- Write a `function` (with a `for` loop inside) that performs a simple population simulation using the following models:
  - Exponential growth: $n_t = n_{t-1}(1 + r)$

Hint: functions need input variables that tell them what to do. The input variables here could be things like *starting population* (n_0), *growth rate* (r), and *number of time steps* (T) to simulate

Second challenge

Population growth models are common in ecology, and usually often take the form $n_t = n_{t-1} + rn_{t-1}$, where n is the number of critters at some time point t, and r is the change in n from one point to the next (r = 0: no change).

- Write a `function` (with a `for` loop inside) that performs a simple population simulation using the following models:
  - Exponential growth: $n_t = n_{t-1}(1 + r)$
  - Logistic growth: $n_t = n_{t-1}(1 + r(1 - \frac{n_{t-1}}{k}))$

Hint: functions need input variables that tell them what to do. The input variables here could be things like *starting population* (n_0), *growth rate* (r), and *number of time steps* (T) to simulate

Second challenge

Population growth models are common in ecology, and usually often take the form $n_t = n_{t-1} + rn_{t-1}$, where n is the number of critters at some time point t, and r is the change in n from one point to the next (r = 0: no change).

- Write a `function` (with a `for` loop inside) that performs a simple population simulation using the following models:
  - Exponential growth: $n_t = n_{t-1}(1 + r)$
  - Logistic growth: $n_t = n_{t-1}(1 + r(1 - \frac{n_{t-1}}{k}))$
- Make a simple plot of your results

Hint: functions need input variables that tell them what to do. The input variables here could be things like *starting population* (n_0), *growth rate* (r), and *number of time steps* (T) to simulate

Lists

- Lists look similar to vectors, but can hold anything in each slot, including other lists.
- LOTS of things in R (e.g. model output) are specially-structured lists at their core

```
myList <- list(charSlot = myCharVec,
    numSlot = myNumVec, logSlot = myLogVec,
    dfSlot = myDF)
```

```
## $charSlot
## [1] "Parakeets" "Cats"      "Dogs"
##
## $numSlot
## [1] 1 2 3 4 5
##
## $logSlot
## [1]  TRUE  TRUE FALSE  TRUE FALSE FALSE
##
## $dfSlot
##   stringCol numCol logCol
## 1 Parakeets      1   TRUE
## 2      Cats      2   TRUE
## 3      Dogs      3  FALSE
```

Accessing Lists

Lists can be accessed numerically or logically, or by their name slots:

```
myList[[2]]  #Needs 2 square brackets to isolate object
```

```
## [1] 1 2 3 4 5
```

```
myList[["numSlot"]]
```

```
## [1] 1 2 3 4 5
```

```
myList$numSlot
```

```
## [1] 1 2 3 4 5
```

```
myList[[4]][, 3]  #Same as myList$dfSlot$logCol
```

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

lapply and *sapply*

Loops can be slow: using a **functional** is generally better (if each step doesn't depend on the previous one)

lapply and *sapply*

Loops can be slow: using a **functional** is generally better (if each step doesn't depend on the previous one)

```r
input <- c(1, 2, 3)  #Input vector
pFun <- function(N, type = "normal") {
    if (type == "poisson")
        rpois(N, 3) else rnorm(N, 3)
}
lapply(input, pFun, type = "poisson")  #Generates a list of N Poisson random numbers
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] 4 2
##
## [[3]]
## [1] 2 0 4
```

lapply and *sapply*

Loops can be slow: using a **functional** is generally better (if each step doesn't depend on the previous one)

```r
input <- c(1, 2, 3)  #Input vector
pFun <- function(N, type = "normal") {
    if (type == "poisson")
        rpois(N, 3) else rnorm(N, 3)
}
lapply(input, pFun, type = "poisson")  #Generates a list of N Poisson random numbers
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] 4 2
##
## [[3]]
## [1] 2 0 4
```

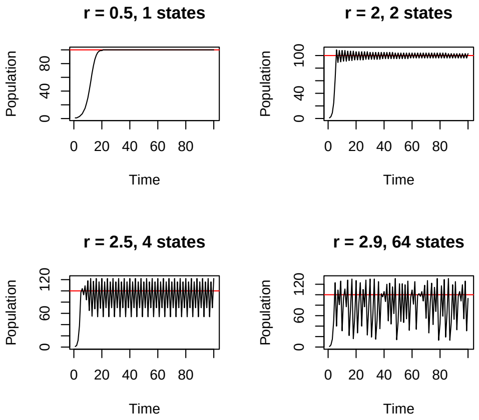
Different output formats: *lapply* uses a list, *sapply* uses a vector (or matrix)

Third challenge

The discrete logistic growth equation has some interesting mathematical properties (see here). At certain values of r the population converges to the carrying capacity K, while at other values it cycles between multiple values.

Using *lapply*, *sapply* and the logistic function you wrote for the second challenge:

- Write a script that iterates across a range of r values: $1 \leq r \leq 3$
- At each r value, how many unique values exist?
- Keep K at 100 and n_0 at 1 for now. Use a simulation time of $t = 1000$, but discard the first 100 points



To do this week:

Get a dataset to work with! This can be:

1. Your own dataset that you want to analyze

Try loading this data into R and play around with it: make some simple plots, or get some summary statistics. Bring it next week and we'll continue using it.

To do this week:

Get a dataset to work with! This can be:

1. Your own dataset that you want to analyze
2. Another dataset that you think might look like your future one

Try loading this data into R and play around with it: make some simple plots, or get some summary statistics. Bring it next week and we'll continue using it.

To do this week:

Get a dataset to work with! This can be:

1. Your own dataset that you want to analyze
2. Another dataset that you think might look like your future one

- Can be from another student (ask your supervisor if they have any “leftovers” from previous students)

Try loading this data into R and play around with it: make some simple plots, or get some summary statistics. Bring it next week and we’ll continue using it.

To do this week:

Get a dataset to work with! This can be:

1. Your own dataset that you want to analyze
2. Another dataset that you think might look like your future one

- Can be from another student (ask your supervisor if they have any "leftovers" from previous students)
- Take a look at a data repository like Dryad

Try loading this data into R and play around with it: make some simple plots, or get some summary statistics. Bring it next week and we'll continue using it.