SIRE516

Week 2: Data handling

Indexing

Accessing data container

- Accessing data in the data container (e.g. vector, matrix, data frame and list) is critical
- You must understand how to access data in the container to use R to the full potential
- Please pay your best attentions

General concepts

- Square brackets, '[ ]', are used for indexing (i.e. accessing data in the containers)
- Notes to programmers
  - The first position of the member in the container is **'1' in R not '0'**
  - '-n' is used to exclude the member at position 'n', not for accessing the n^{th} member counting from the last position

Vector

- Know your vector

```
1  ####Accessing vector pt 1####
2  letters #'letters' is a built-in vector containing lower case alphabets
3
4  print(letters) #Show everything in the vector
5
6  length(letters) #Number of members
7
8  class(letters) #Type of data in the vector
9  typeof(letters) #Type of data in the vector
10
11 summary(letters) #Summary of the vector
12 str(letters) #Structure of the vector
13
```

Vector

- Accessing by number

```
####Accessing vector pt 2####
letters[1] #First member
letters[20] #Twentieth member
letters[30] #Try and see what happens

letters[1:10] #First to tenth
letters[c(1,10)] #First AND tenth
letters[c(1,10,20)] #First, tenth and Twentieth
letters[c(1:10,20)] #First to tenth and Twentieth
letters[seq(from = 1, to = 26, by =2)] #Every other members

letters[-26] #Drop the last member
length(letters)
letters[-length(letters)] #Drop the last member
letters[-seq(from = 1, to = 26, by =2)]

#Drop the last 7 members of 'letters' vector
#letters[???]
```

Vector

- Accessing by condition/logic

```
####Accessing vector pt 3####
vowel <- c("a","e","i","o","u")
letters #Built-in vector for lower case alphabets

flags <- which(letters %in% vowel)
letters[flags] #Flagged positions
letters[-flags] #Un-flagged positions

#'%in%' tests each member in the 'letters' vector whether it is "in"
#the 'vowel' vector
letters %in% vowel
#'which' returns the position of TRUE
which(letters %in% vowel)

#Advanced user; Code might be more confusing
letters[which(letters %in% vowel)]
letters[which(!(letters %in% vowel))]
letters[-which(letters %in% vowel)]
```

Vector

- Accessing by condition/logic

```
####Accessing vector pt 4####
a <- runif(n = 100,min = 1,max = 10)
#Randomly generate 100 real numbers greater than 1, but smaller than 10

flags <- which(a >= 9)
a[flags]
a[-flags]

#Advanced user; Code might be more confusing
a[a>=9]
a[!(a>=9)]

#Why use "which"? NA and NaN
b <- c(a,NA,0/0)
print(b)
flags <- which(b >= 9)
b[flags]
b[b >= 9] #Look at the last two members
```

Matrix

- Know your matrix

```
71  ####Accessing matrix pt 1####
72  a <- runif(n = 12,min = 1,max = 10)
73  am <- matrix(data = a, nrow = 4)
74
75  print(am) #Show everything in the matrix
76
77  dim(am) #Dimension of the matrix; row column
78  nrow(am) #Number of rows
79  ncol(am) #Nuber of columns
80
81  class(am) #Container type
82  typeof(am) #Type of data in the matrix
83
84  summary(am) #Summary of the matrix
85  str(am) #Structure of the matrix
```

Matrix

- Accessing by number

```
87  ####Accessing matrix pt 2####
88  a <- runif(n = 12,min = 1,max = 10)
89  am <- matrix(data = a, nrow = 4)
90
91  print(am)
92  #Matrix[Row,Column]
93  am[1, ] #Row 1; 'All' columns -> Vector
94  am[ ,1] #All rows; Column 1 -> Vector
95  am[1,1] #Row 1, Column 1 -> Cell/Single value
96
97  am[c(1,2,4), c(1,3)] #Row 1, 2 and 4; Column 1 and 3
98  am[-3,-2] #Drop row 3; Drop column 2 (i.e. H4 x W3 -> H3 x W2)
99
100 am[nrow(am),ncol(am)] #Last row, Last column -> Cell/Single value
101
102 #Row 2 to 4, Column1
103 #am[???]
```

Data frame

- Know your data frame

```
105 ####Accessing data frame pt 1####
106 ?mtcars #'mtcars' is a built-in data frame for 'Motor Trend Car Road Tests'
107
108 print(mtcars) #Show everything in the data frame
109 View(mtcars) #Nice way to look at the data frame
110
111 dim(mtcars) #Dimension of the data frame; row column
112 nrow(mtcars) #Number of rows
113 ncol(mtcars) #Nuber of columns
114
115 class(mtcars) #Container type
116 typeof(mtcars) #Container type
117
118 summary(mtcars) #Summary of the data frame
119 str(mtcars) #Structure of the data frame
```

Data frame

- Accessing by name

```
120 ####Accessing data frame pt 2####
121 names(mtcars) #Names in the data frame
122 colnames(mtcars) #Names of columns
123 rownames(mtcars) #Names of rows
124 
125 #Call a column by name
126 mtcars$mpg #All data in column 'mpg' -> vector
127 mtcars["mpg"]#All data in column 'mpg' -> data.frame
128 mtcars[["mpg"]]  #All data in column 'mpg' -> vector
129 mtcars[ ,"mpg"] #All row; Column 'mpg' -> vector
130 
131 #Call a row by name
132 mtcars["Mazda RX4",]
133 
134 #Call a raw and columns by names
135 mtcars["Mazda RX4",c("mpg","hp","gear")]
```

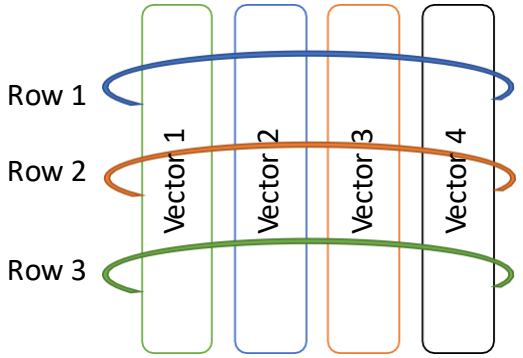
Data frame

- Accessing by number

```
138 ####Accessing data frame pt 3####
139 mtcars[10,] #Tenth row
140 mtcars[,10] #Tenth column == mtcars$gear
141 mtcars[10,10] #Tenth row; Tenth column
142
143 #Mixed with "names"
144 mtcars$mpg[1:10] #Column 'mpg'; Row 1 to 10 -> vector
145 mtcars[1:10,"mpg"] #Row 1 to 10, Column 'mpg' -> vector
146 mtcars[1:10,]$mpg #Row 1 to 10, Column 'mpg' -> vector
147
148 mtcars[c(1,10,12), c("mpg","gear")] #Row 1, 10 and 12; Column 'mpg' and 'gear'
```

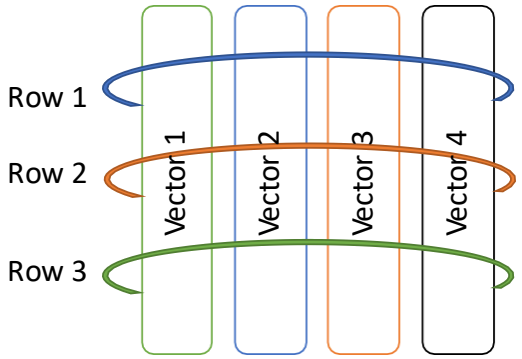
Data frame

- Accessing by condition/logic
- Imagine multiple vectors with identical lengths bound together into “rows”
- Member 1 of Column(vector) 1 is linked to Member 1 of Column 2, Member 1 of Column 3, …,and Member 1 of Column N = Row 1



Data frame

- Accessing by condition/logic
- Apply logic/condition to one column to get the positions of members of that column
- Use the positions of members in this column to get the member in other columns
- In other words, use the positions of members in this column to get the rows to which these members belong



Data frame

- Accessing by condition/logic

```
149 ####Accessing data frame pt 4####
150 flags <- which(mtcars$gear < 4) #Which row has gear < 4
151 
152 print(mtcars$gear < 4)
153 print(which(mtcars$gear < 4))
154 
155 mtcars[flags, ] #Use row number to select row; all columns -> data frame
156 mtcars[flags, ]$mpg  #Use row number to select row; Column 'mpg' -> vector
157 mtcars[flags,"mpg"]  #Use row number to select row; Column 'mpg' -> vector
158 mtcars$mpg[flags] #Column 'mpg'; Use row number to select row -> vector
159 mtcars[flags,c("mpg","hp")]  #Use row number to select row; Column 'mpg' and 'hp'
160 
161 flags <- which(mtcars$gear < 4 & mtcars$hp < 100)
162 #Which row has gear < 4 and hp < 100
163 mtcars[flags, ]
164 #Advanced user
165 mtcars[ which(mtcars$gear < 4 & mtcars$hp < 100),]
```

List

- Know your list

```
167 ####Accessing list pt 1####
168 A <- list(a = c(T,F), b = 1:6, c = letters, d = mtcars)
169
170 print(A) #Show everything in the list
171
172 class(A) #Container type
173 typeof(A) #Container type
174
175 summary(A) #Summary of the list
176 str(A) #Structure of the list (More useful)
```

List

- Accessing by number

```
178 ####Accessing list pt 2####
179 A[1] #Member 1 of the list -> list
180 A[[1]] #Member 1 of the list -> vector
181
182 A[[1]][2] #Member 2 of Member 1 of the list
183 A[1][2] #Try and see what happens
```

List

- Accessing by name

```
185 ####Accessing list pt 3####
186 names(A) #Names in list A
187 str(A) #Structure of list A
188 
189 A$a #Return vector
190 A["a"] #Return list (Not useful)
191 A[["a"]] #Return vector
192 
193 A["d"] #Return list (Not useful)
194 A$d #Return data frame
195 A$d$mpg #Access column of the data frame in the list
```

Practical 1

Practical 1: Exploring data frame

- We will explore “hflights” dataset

1. Find number of columns
2. Find number of rows
3. How many flights were flying to "JFK" airport (Hint: "Dest" is 'JFK’)
4. How many flights were flying to "DTW" airport on January 31, 2011
5. What airlines were flying to "DTW" airport on January 31, 2011 (Hint:'UniqueCarrier')

```
1  library(hflights)
2
3  ?hflights #Description of the dataset
```

Practical 2

Practical 2: Exploring regression results

- Summary of regression results is in a data container
- There are many components in the summary
- We might need extract a few information from the summary
- We will explore the regression results of 'ChickWeight' dataset

```
?ChickWeight #Description of the dataset

####Do not worry about regression, yet. Just run the commands
model1 <- lm(weight ~ Time , data = ChickWeight) #Fitting model
sum1 <- summary(model1) #Summarize the fitting results

print(sum1)
```

Practical 2: Exploring regression results

1. What is the type of data container of 'sum1' (i.e. data frame or list)?
2. How many components are in "sum1"?
3. Extract "r.squared”
4. Extract p-value (i.e. Pr(>|t|)) of 'Time' (Hint: 'coefficients')

```r
?ChickWeight #Description of the dataset

####Do not worry about regression, yet. Just run the commands
model1 <- lm(weight ~ Time , data = ChickWeight) #Fitting model
sum1 <- summary(model1) #Summarize the fitting results

print(sum1)
```

Import and Export Data

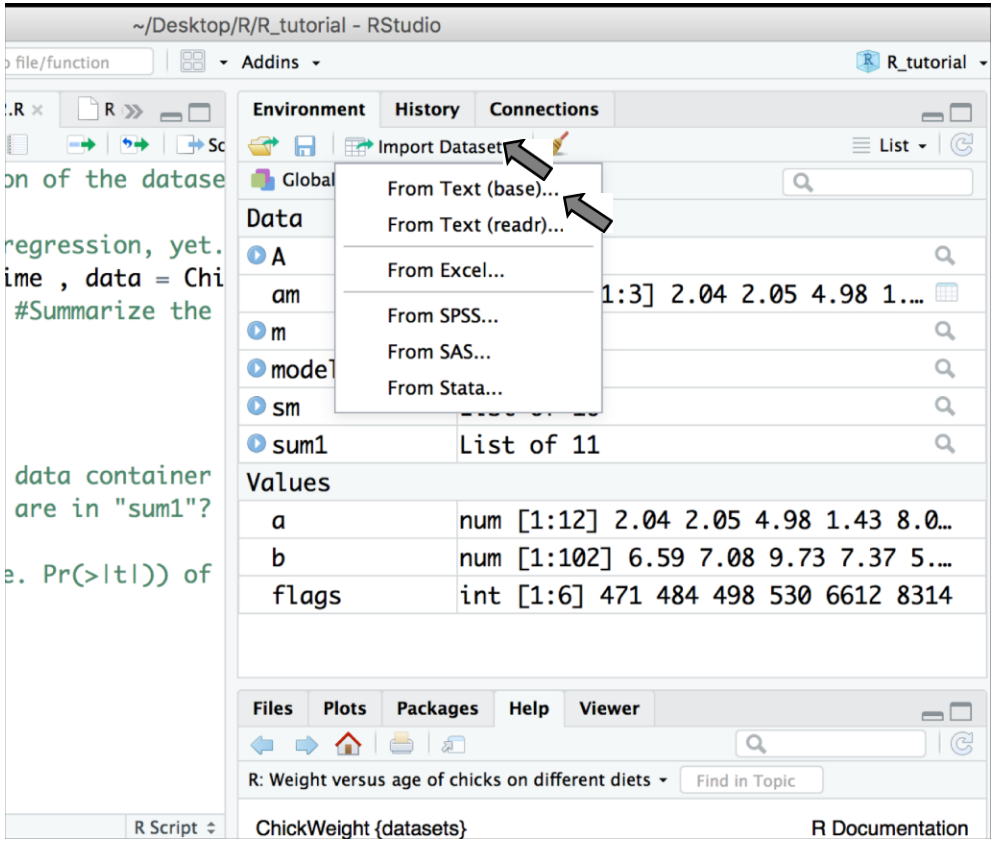
Data formats

- **Text file: e.g. CSV and Tab delimited**
- **MS Excel**
- Other statistical software: SPSS, SAS and STATA
- Database: e.g. SQL, MS Access and noSQL
- Website

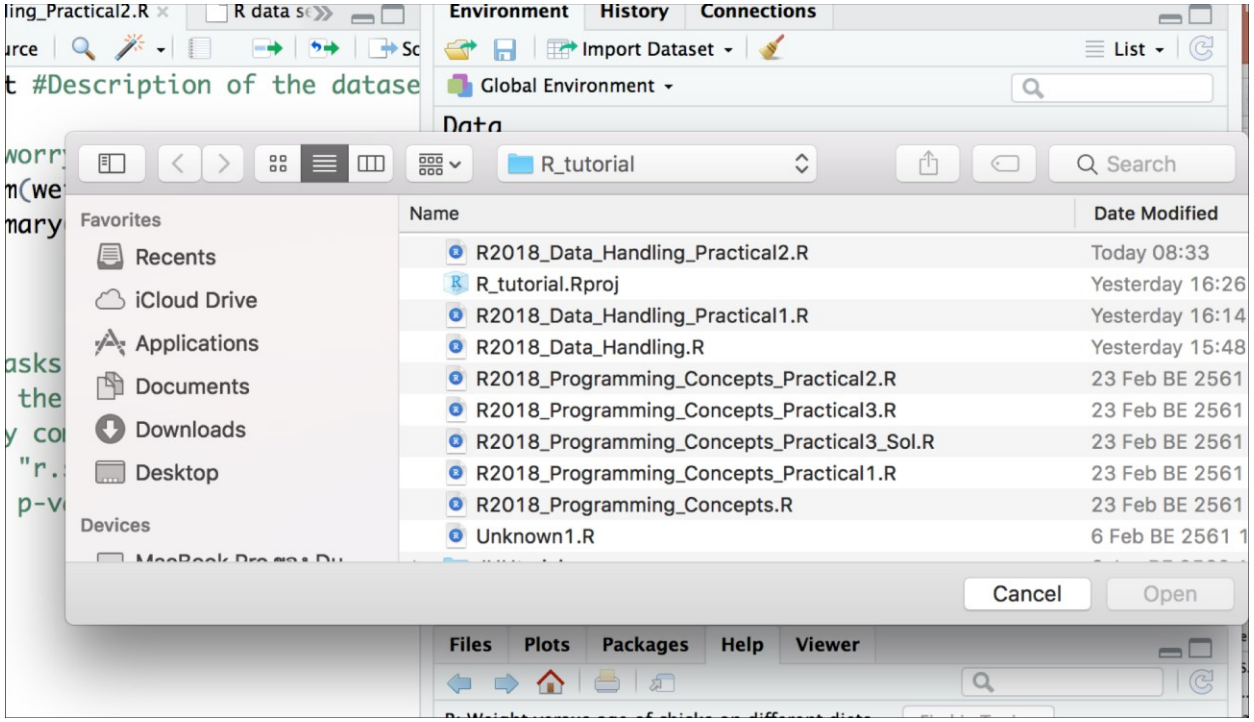
Text files

- The simplest format for importing/exporting
- Unlikely to generate an error during importing/exporting
- The current version of RStudio provides interfaces helping import data

Text files

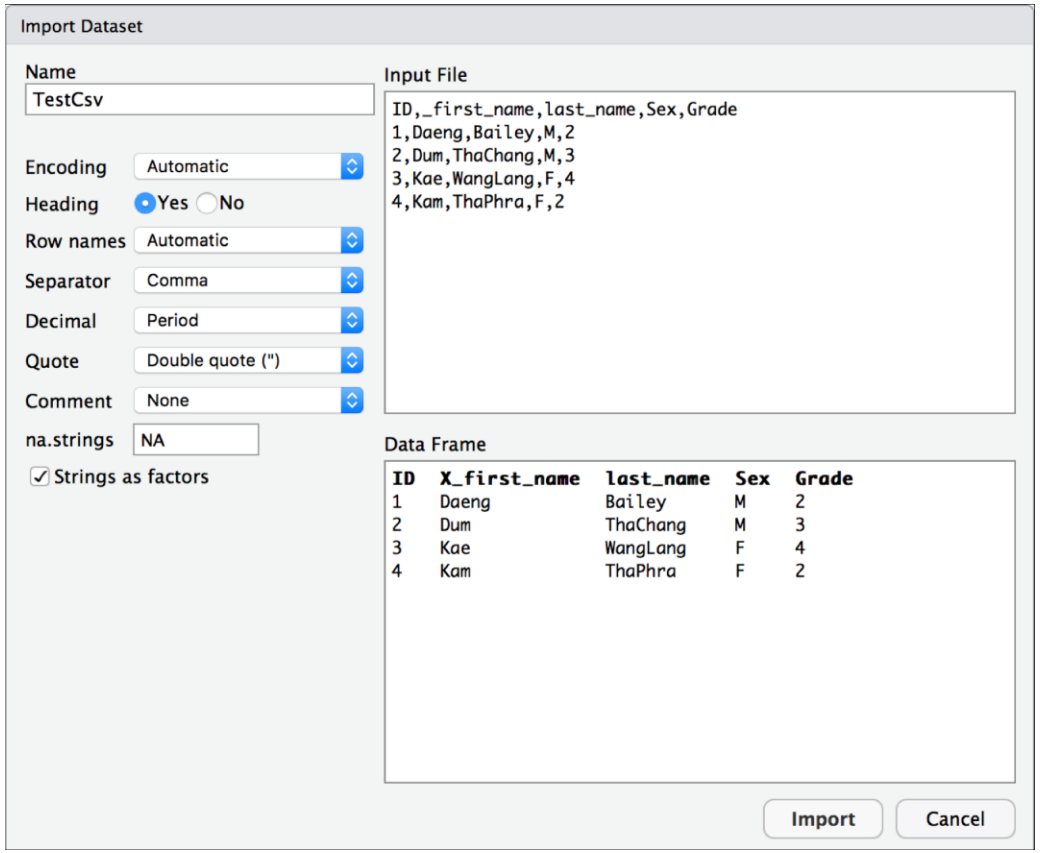

~/Desktop/R/R_tutorial - RStudio
Addins
R_tutorial
Environment
History
Connections
Import Dataset
List
From Text (base)...
From Text (readr)...
From Excel...
From SPSS...
From SAS...
From Stata...
Data
A
am
m
model
sm
sum1
List of 11
Values
a
num [1:12] 2.04 2.05 4.98 1.43 8.0…
b
num [1:102] 6.59 7.08 9.73 7.37 5.…
flags
int [1:6] 471 484 498 530 6612 8314
Files
Plots
Packages
Help
Viewer
R: Weight versus age of chicks on different diets
Find in Topic
ChickWeight {datasets}
R Documentation
R Script

Text files



Choose your file

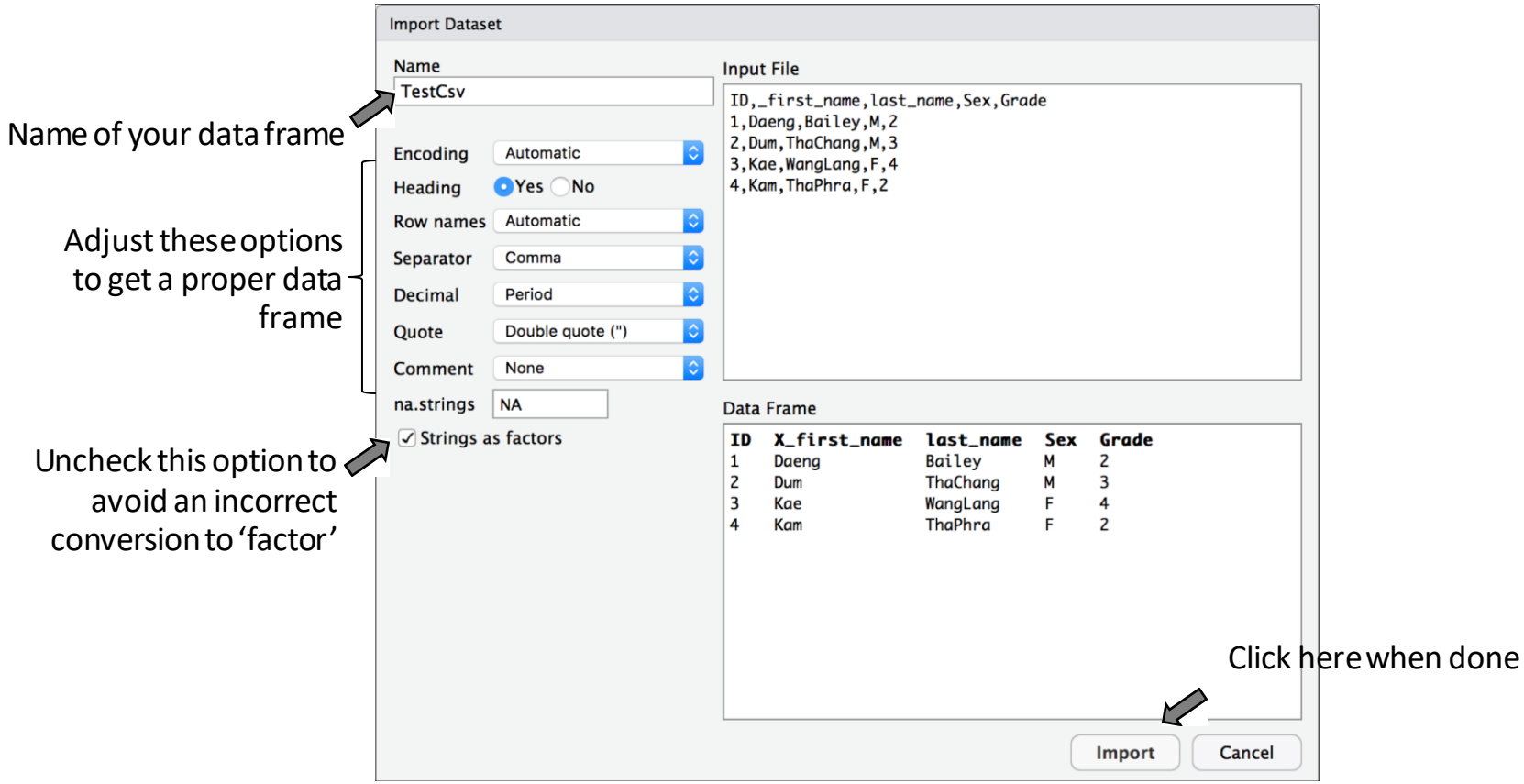
Text files



RAW input

Preview imported data

Text files


Name of your data frame
Adjust these options to get a proper data frame
Uncheck this option to avoid an incorrect conversion to 'factor'
Import Dataset
Name
TestCsv
Encoding
Automatic
Heading
Yes
No
Row names
Automatic
Separator
Comma
Decimal
Period
Quote
Double quote (")
Comment
None
na.strings
NA
Strings as factors
Input File
ID,_first_name,last_name,Sex,Grade
1,Daeng,Bailey,M,2
2,Dum,ThaChang,M,3
3,Kae,WangLang,F,4
4,Kam,ThaPhra,F,2
Data Frame
ID X_first_name last_name Sex Grade
1 Daeng Bailey M 2
2 Dum ThaChang M 3
3 Kae WangLang F 4
4 Kam ThaPhra F 2
Click here when done
Import
Cancel

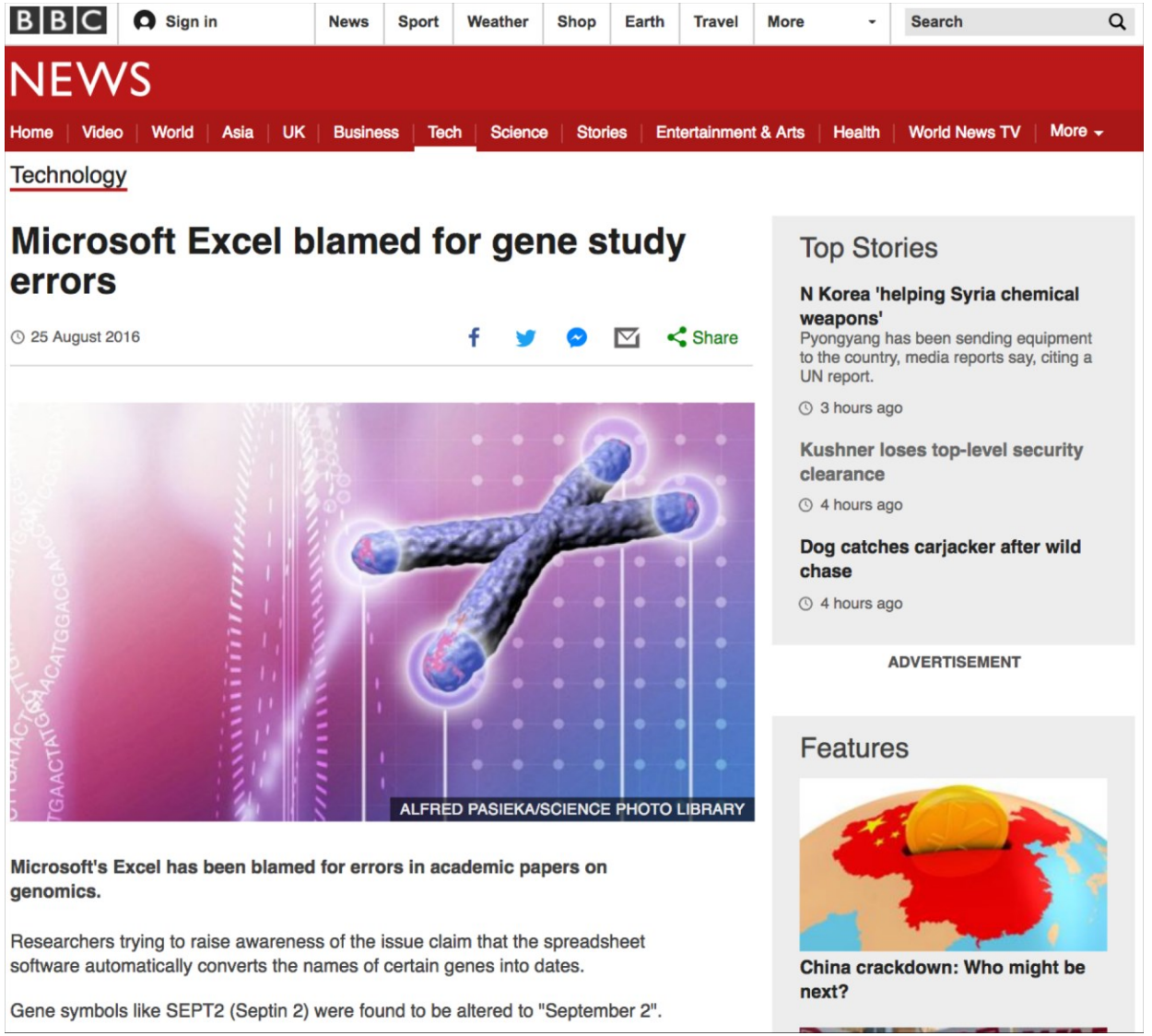
Text files

- By clicking “Import,” RStudio will auto-generate an importing command in the console panel
- The text file will be imported and shown in the Viewer
- **BEST Practice:** the auto-generated command for importing data should be copied from the console panel and included in your R code
  - Otherwise, you will have to browser the file again if the R code must be re-run

Excel files

- Frequently used format
- MS Excel has a lot of tools for data preparation

Excel files



BBC Sign in News Sport Weather Shop Earth Travel More Search

NEWS

Home Video World Asia UK Business Tech Science Stories Entertainment & Arts Health World News TV More

Technology

Microsoft Excel blamed for gene study errors

25 August 2016 Share

ALFRED PASIEKA/SCIENCE PHOTO LIBRARY

Microsoft's Excel has been blamed for errors in academic papers on genomics.

Researchers trying to raise awareness of the issue claim that the spreadsheet software automatically converts the names of certain genes into dates.

Gene symbols like SEPT2 (Septin 2) were found to be altered to "September 2".

Top Stories

N Korea 'helping Syria chemical weapons'
Pyongyang has been sending equipment to the country, media reports say, citing a UN report.
3 hours ago

Kushner loses top-level security clearance
4 hours ago

Dog catches carjacker after wild chase
4 hours ago

ADVERTISEMENT

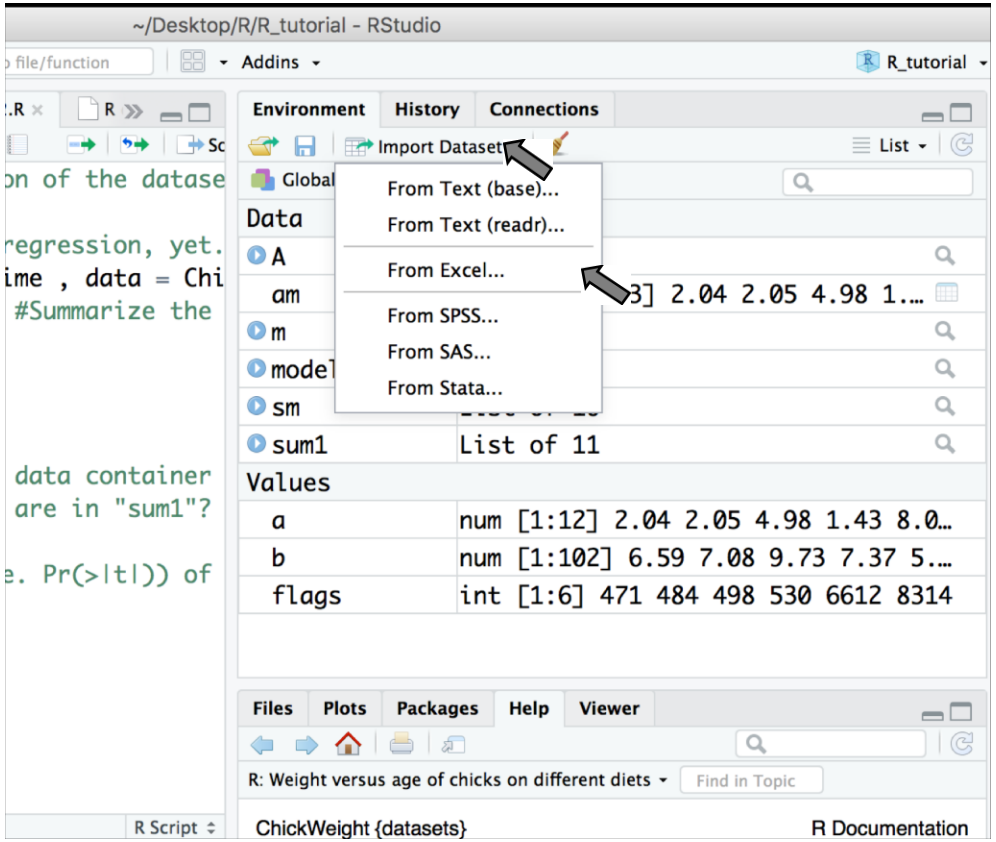
Features

China crackdown: Who might be next?

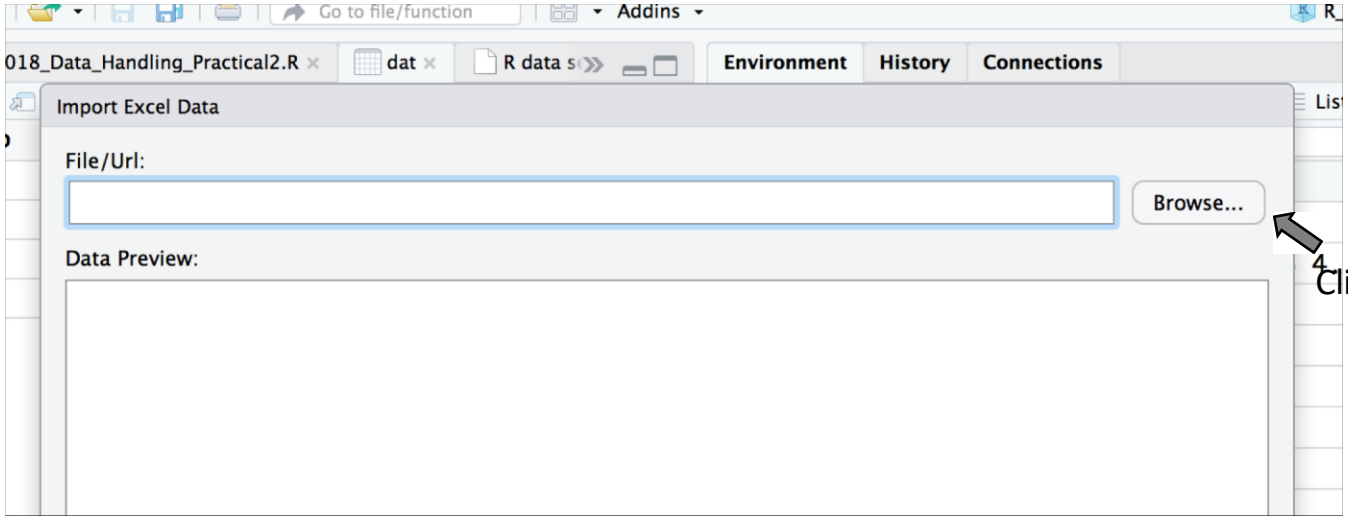
Excel files

- **CAUTION 1:** Excel's automatic data conversion can generate critical errors
- **CAUTION 2:** Avoid opening and saving your CSV and TAB files in MS Excel
- If you need to import the data from Excel files
  - If possible, convert the file to CSV before importing
  - Thoroughly check imported data

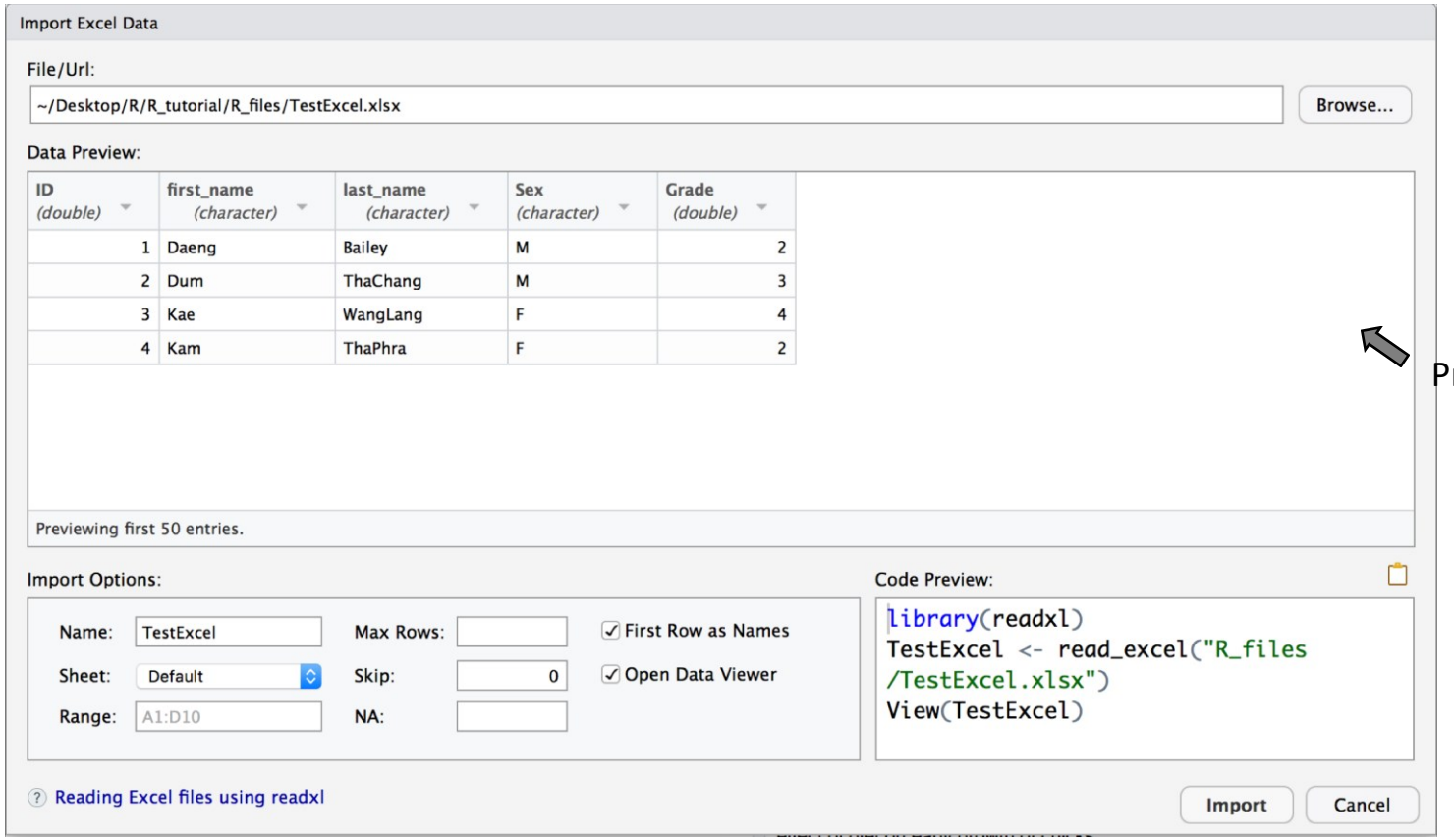
Excel files


~/Desktop/R/R_tutorial - RStudio
Addins
R_tutorial
Environment
History
Connections
Import Dataset
List
From Text (base)...
From Text (readr)...
From Excel...
From SPSS...
From SAS...
From Stata...
Data
A
am
m
model
sm
sum1
List of 11
Values
a
num [1:12] 2.04 2.05 4.98 1.43 8.0…
b
num [1:102] 6.59 7.08 9.73 7.37 5.…
flags
int [1:6] 471 484 498 530 6612 8314
Files
Plots
Packages
Help
Viewer
R: Weight versus age of chicks on different diets
Find in Topic
ChickWeight {datasets}
R Documentation
R Script

Excel files

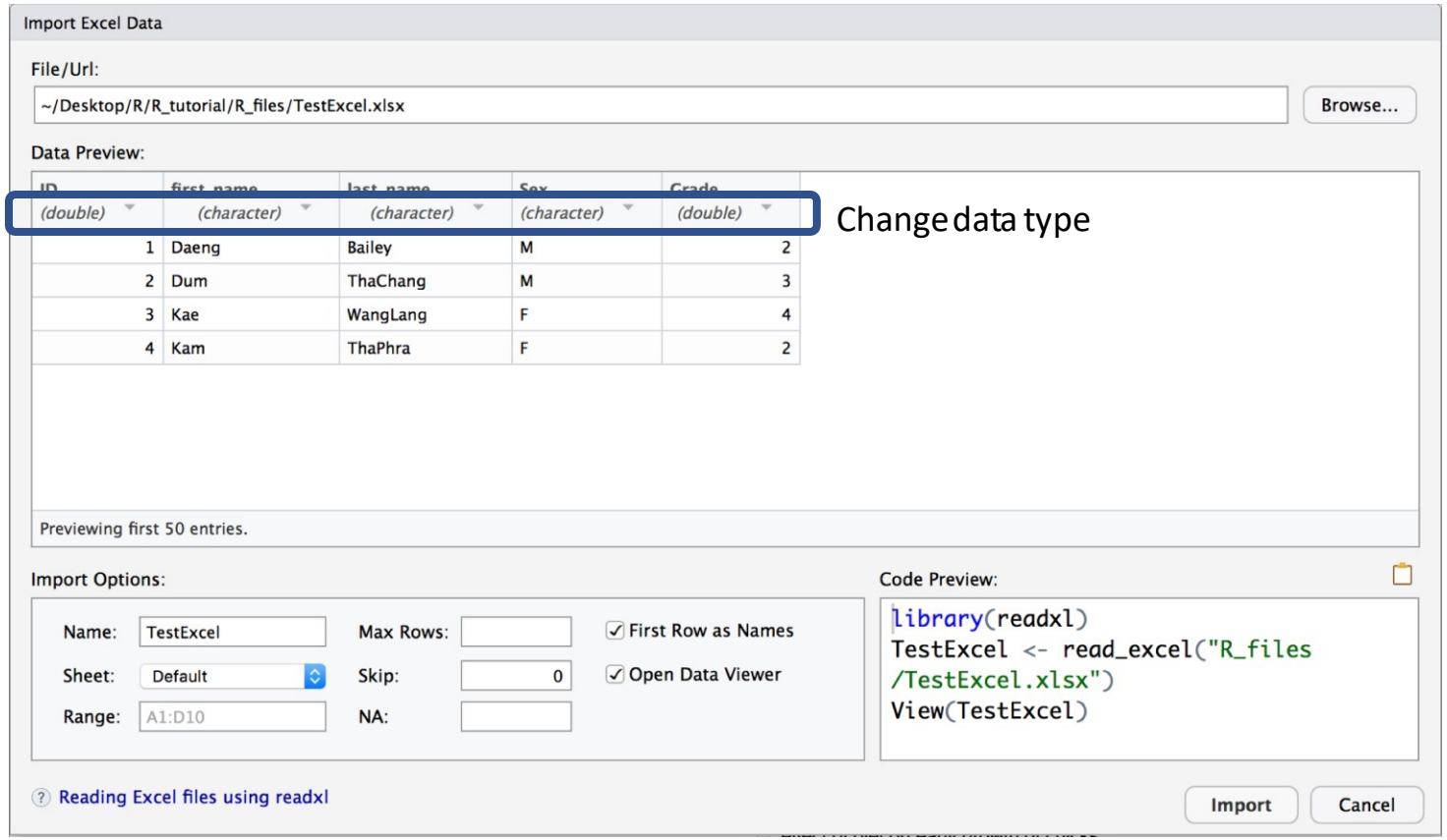

Go to file/function
Addins
018_Data_Handling_Practical2.R
dat
R data s
Environment
History
Connections
Import Excel Data
File/Url:
Browse...
Data Preview:
4
Click and select your file

Excel files

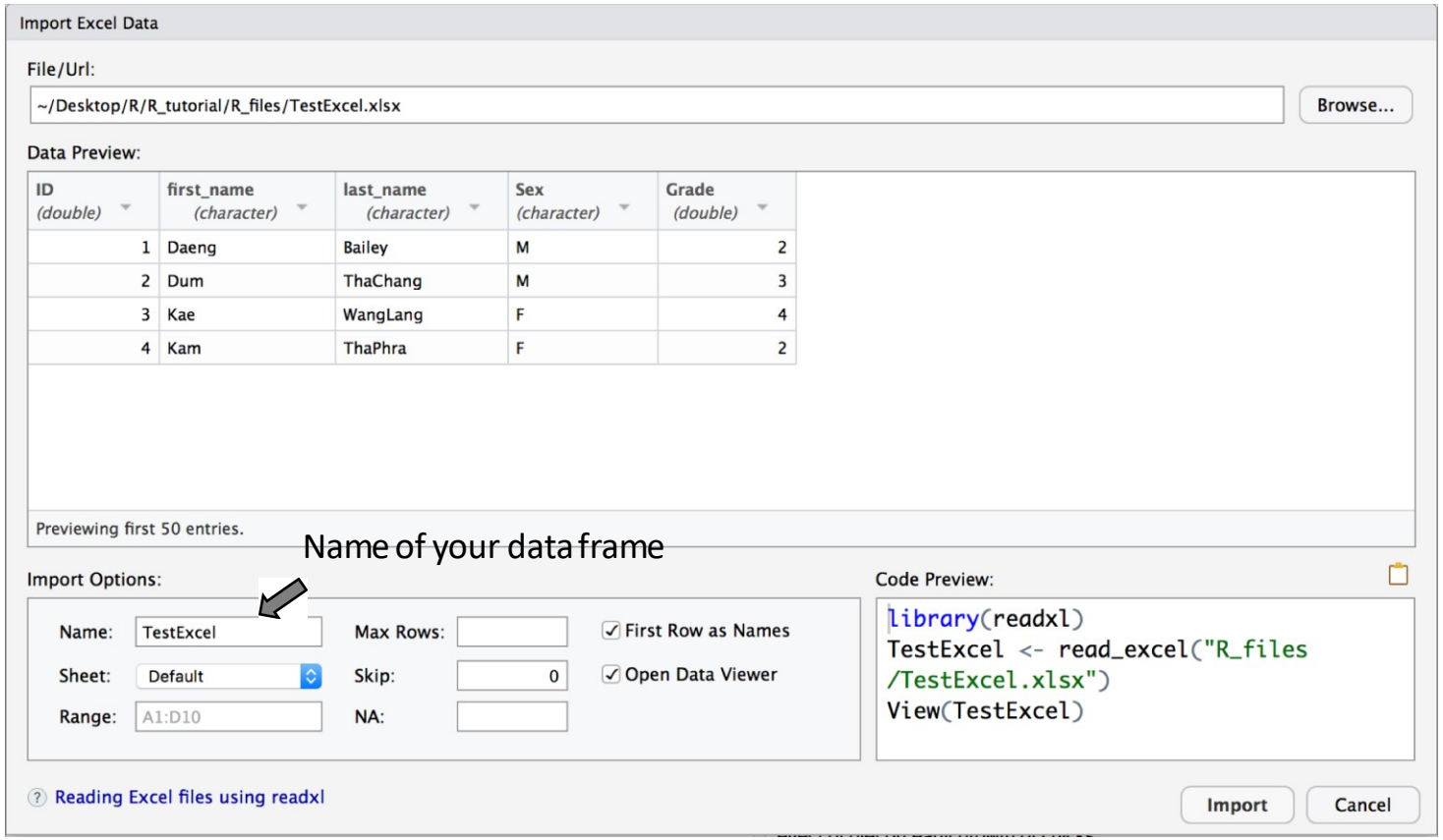


Preview of imported data

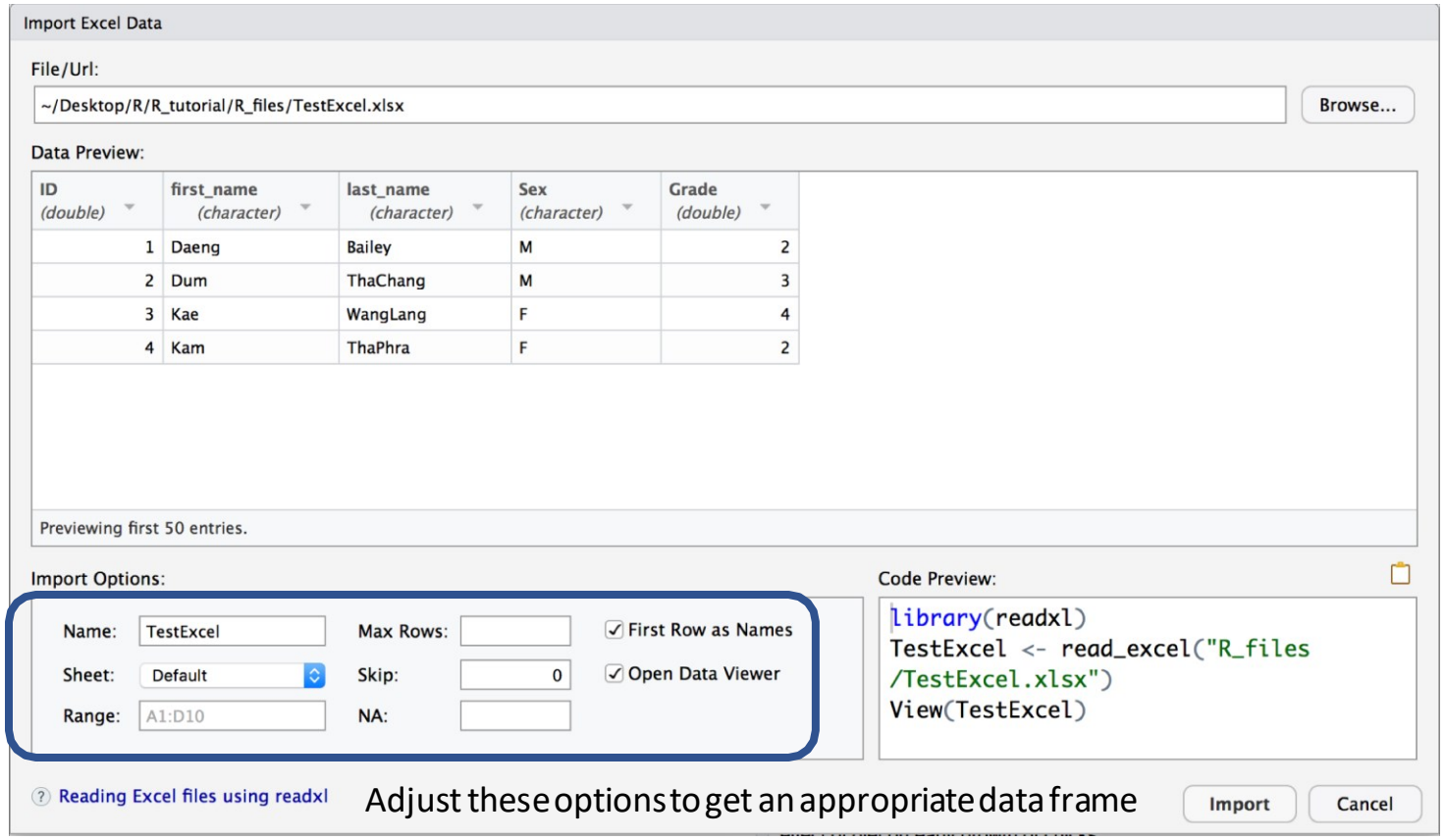
Excel files


Import Excel Data
File/Url:
~/Desktop/R/R_tutorial/R_files/TestExcel.xlsx
Browse...
Data Preview:
ID (double)
first_name (character)
last_name (character)
Sex (character)
Grade (double)
1 Daeng Bailey M 2
2 Dum ThaChang M 3
3 Kae WangLang F 4
4 Kam ThaPhra F 2
Previewing first 50 entries.
Change data type
Import Options:
Name: TestExcel
Sheet: Default
Range: A1:D10
Max Rows:
Skip: 0
NA:
First Row as Names
Open Data Viewer
Code Preview:
library(readxl)
TestExcel <- read_excel("R_files
/TestExcel.xlsx")
View(TestExcel)
Reading Excel files using readxl
Import
Cancel

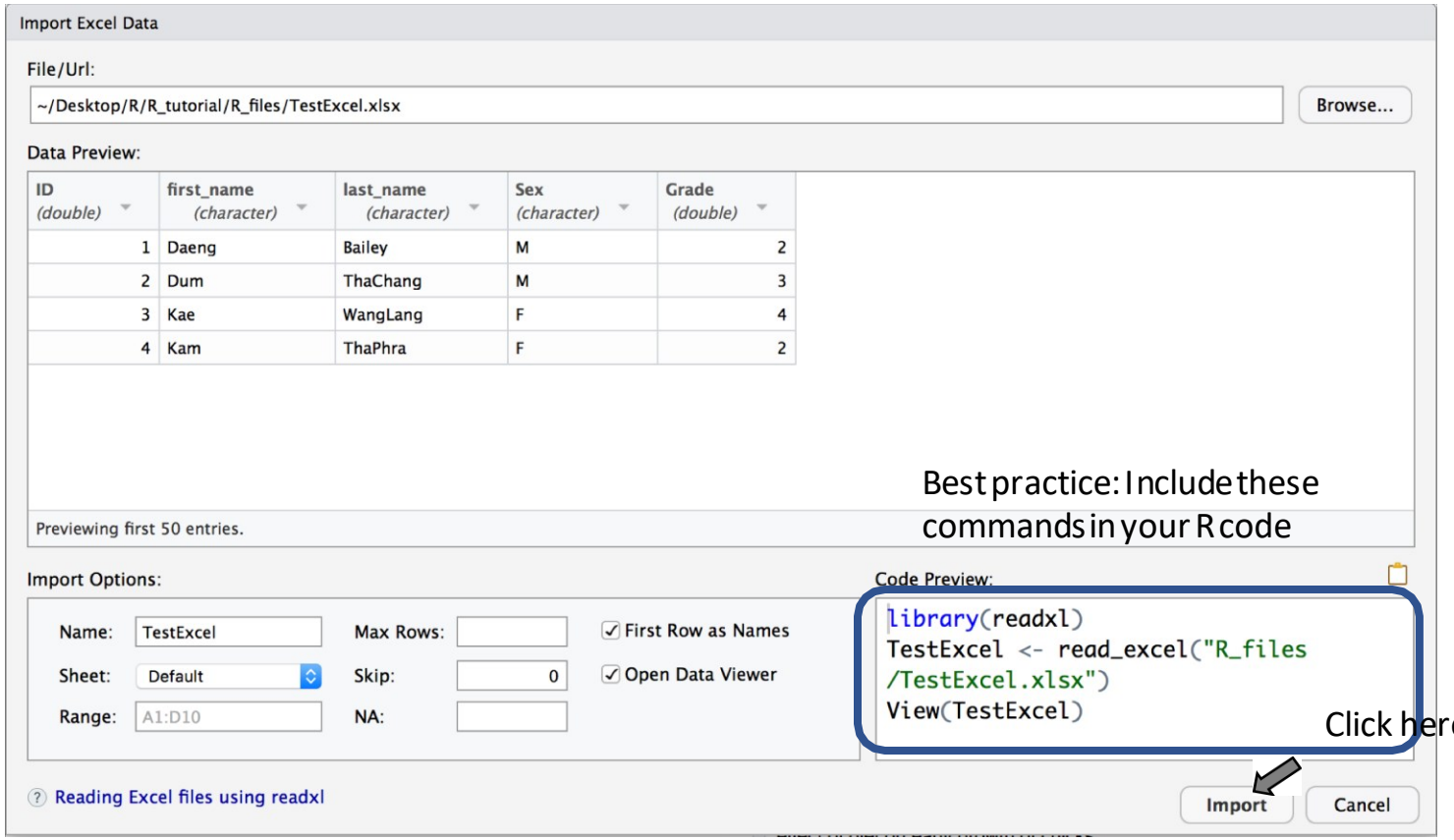
Excel files


Import Excel Data
File/Url:
~/Desktop/R/R_tutorial/R_files/TestExcel.xlsx
Browse...
Data Preview:
ID (double)
first_name (character)
last_name (character)
Sex (character)
Grade (double)
1 Daeng Bailey M 2
2 Dum ThaChang M 3
3 Kae WangLang F 4
4 Kam ThaPhra F 2
Previewing first 50 entries.
Name of your data frame
Import Options:
Name: TestExcel
Sheet: Default
Range: A1:D10
Max Rows:
Skip: 0
NA:
First Row as Names
Open Data Viewer
Code Preview:
library(readxl)
TestExcel <- read_excel("R_files
/TestExcel.xlsx")
View(TestExcel)
Reading Excel files using readxl
Import
Cancel

Excel files


Import Excel Data
File/Url:
~/Desktop/R/R_tutorial/R_files/TestExcel.xlsx
Browse...
Data Preview:
ID (double)
first_name (character)
last_name (character)
Sex (character)
Grade (double)
1 Daeng Bailey M 2
2 Dum ThaChang M 3
3 Kae WangLang F 4
4 Kam ThaPhra F 2
Previewing first 50 entries.
Import Options:
Name: TestExcel
Sheet: Default
Range: A1:D10
Max Rows:
Skip: 0
NA:
First Row as Names
Open Data Viewer
Code Preview:
library(readxl)
TestExcel <- read_excel("R_files
/TestExcel.xlsx")
View(TestExcel)
Reading Excel files using readxl
Adjust these options to get an appropriate data frame
Import
Cancel

Excel files

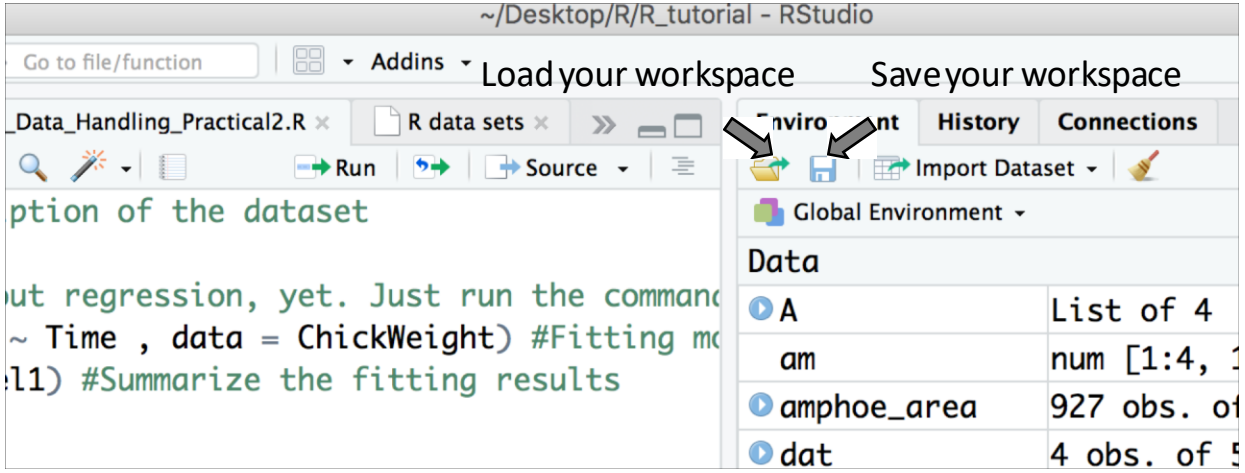


Best practice: Include these commands in your R code

Click here when done

Save/Export your data

- Save your “workspace” as .Rdata file
- The simplest method
- Your data frame will looked the same when reloaded
- Your data will **not** be accessible by text editor or Excel



Save/Export your data

- Save your data as a text file (e.g. CSV, TAB)
- Your data will be accessible by text editor and Excel
- Your data can be later converted to Excel file

```
####Saving your data####
?mtcars #Description of the data frame
?write.csv #Description of the command for saving the data

write.csv(x = mtcars, #Specify your data.frame to be saved
  file = "~/Desktop/R/mtcars.csv", #Specify the file name and path
  na = '', #OPTIONAL: How will the missing data be shown (default = 'NA')
  row.names = FALSE #OPTIONAL: Show row.names or not (default = TRUE)
  )

write.csv(x = mtcars,
          file = "~/Desktop/R/mtcars2.csv"
)
```

Other ways to import/export data

- Package ‘xlsx’ -> Save/Load your data as excel
- Package ‘RMySQL’ -> Interact with SQL database
- Package ‘haven’ -> Save/Load your data in SAS, SPSS and STATA formats
- Package ‘RCurl’ -> Interact with web application

Practical 3

Practical 3: Import your data

- Create a table in MS Excel with the first two rows of data as follows

	A	B	C	D	E	F	G
1	ID	First_Name	Last_Name	Birth_Date	Sex	Weight	Height
2	1	John	Doe	12/8/1970	M	75	167
3	2	Jane	Doe	17/12/2000	F	Not Measured	159
4							

- Add at least 2 more rows with any random data
- Save the file as Excel (.xlsx or .xls)
- Save another copy as CSV (.csv in “Save As”)

Practical 3: Import your data

- Import Excel and CSV files into R
- Check if they are imported correctly (summary() and str())
  - Data
  - Data type
- Correct data types
- Save your data as CSV with a new name

Data Frame Operations

FILE:R2018_Data_Handling_pt2.R

Data Frame Operations

- Each column can be used as a variable for calculation
- New column can be assigned to store new variable from the calculation

```r
####Data Frame Operations pt 1####
?women #Description of the data frame
str(women)

#Create new columns with calculated/transformed data from other column
print(women$height_m) #Before assignment
women$height_m <- women$height * 0.0254 #Convert inch to meter
print(women$height_m) #After assignment

women$weight_kg <- women$weight * 0.4536 #Convert pound to kilogram
women$BMI <- women$weight_kg/(women$height_m ^ 2)

print(women)
```

Data Frame Operations

- ‘cbind()’ for adding new column to the data frame

```
####Data Frame Operations pt 2####
nrow(women)
vector1 <- 1:nrow(women)
ID <- data.frame(id = vector1) #vector1 assigned as 'id' column
print(ID)

n_women <- cbind(women,ID)
print(n_women)

#What if number of rows does not match?
vector2 <- 1:10
ID2 <- data.frame(id2 = vector2)
print(ID2)
n_women <- cbind(women,ID2) #Try and see what happens
```

Data Frame Operations

- ‘rbind()’ for adding new row (record) to the data frame

```
####Data Frame Operations pt 3####
data(women) #Load the original women data set
print(women)

new_record <- data.frame(height = 55, weight = 100 )
print(new_record)
new_women <- rbind(women,new_record)
print(new_women)

#What if number of column does not match?
new_record2 <- data.frame( weight = 120 )
print(new_record2)
new_women <- rbind(women,new_record2) #Try and see what happens
```

Data Frame Operations

- ‘merge()’ for joining two data frames
- Similar to ‘JOIN’ in SQL

```
####Data Frame Operations pt 4####
#a = transaction data
a <- data.frame(ID = 1:5,
                buyer = c("DM","PS","AC","KP","YT"),
                seller = c("A","A","B","D","E"))
#b = seller information
b <- data.frame(abbre = c("A","B","C","D"),
                full = c("Apple","Banana","Carrot","Durian"),
                phone = c("111","222","333","444"))
print(a)
print(b)
```

Data Frame Operations

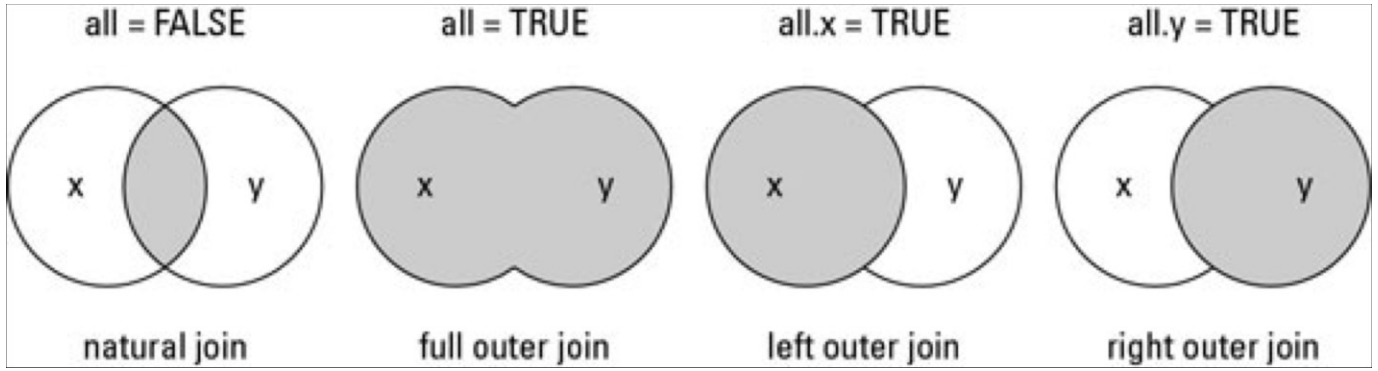
- ‘merge()’ requires:
  - ‘x’ = table 1
  - ‘y’ = table 2
  - ‘by.x’ = names of the column of table 1 to be used for merging
  - ‘by.y’ = names of the column of table 2 to be used for merging
  - ‘all’, ‘all.x’ or ‘all.y’ = whether all rows will be kept with or without matched rows

Data Frame Operations

```
####Data Frame Operations pt 5####
result <- merge(x=a,y=b, by.x = "seller",by.y = "abbre") #Inner join
print(result) #Drop all unmatched

result <- merge(x=a,y=b, by.x = "seller",by.y = "abbre", all.x = T) #Left join
print(result) #Drop unmatched in 'b' (i.e. y or right)

result <- merge(x=a,y=b, by.x = "seller",by.y = "abbre", all.y = T) #Right join
print(result) #Drop unmatched in 'a' (i.e. x or left)

result <- merge(x=a,y=b, by.x = "seller",by.y = "abbre", all = T) #Outer join
print(result) #Do not drop any unmatched

result <- merge(x=a,y=b, by=NULL) #Cross join
print(result) #All combinations
```

Data Frame Operations

- ‘merge()’ for joining two data frames
- Similar to ‘JOIN’ in SQL



http://www.dummies.com/programming/r/how-to-use-the-merge-function-with-data-sets-in-r/

Useful Functions For Data Preparation

Quick look at your data

- View(); **not view()**
- head()
- tail()

```
67  ####Quick look####
68  View(mtcars)
69
70  head(mtcars)
71  head(mtcars, n = 2)
72
73  tail(mtcars)
74  tail(mtcars, n = 3)
```

‘attach()’

- Attach your data frame so that you can call each column directly
- **CAUTION:** If you are working with more than one data frame, referring columns from several attached data frames might be confusing

```
67  ####attach()####
68  ?mtcars
69
70  mtcars$mpg #Call column 'mpg' from 'mtcars' data frame
71  mpg #Try and see what happens
72
73  attach(mtcars)
74  mpg
75  gear
76  hp
77
78  detach(mtcars) #Detach the data frame
79  mpg #Try and see what happens
```

‘unique()’

- Deduplication

```
####unique pt 1####
?mtcars
print(mtcars)

new_mtcars <- rbind(mtcars, mtcars[5:10,])
u_mtcars <- unique(new_mtcars)
print(new_mtcars)
print(u_mtcars) #Deduplicate new_mtcars

dim(mtcars)
dim(new_mtcars)
dim(u_mtcars)

identical(mtcars,new_mtcars) #Check if two objects are identical
identical(mtcars,u_mtcars)
```

‘unique()’ vs ‘levels()’ vs ‘table()’

- Checking for unique values
  - unique(): generic;
  - levels(): for ‘factor’ data
  - table(): tallying up unique values

```
97  ####unique pt 2####
98  ?iris
99
100 iris$Species
101 unique(iris$Species)
102 levels(iris$Species)
103 table(iris$Species)
104
105 c_species <- as.character(iris$Species) #Covert from 'factor' to 'character'
106 levels(c_species) #Try and see what happesn
107 unique(c_species)
108 table(c_species)
```

‘paste()’ and ‘paste0()’

- Combine/Concatenate texts

```
####paste and paste0####
paste("I","love","you.")
paste0("I","love","you.")
paste("I","love","you.", sep = "")

library(hflights)
?hflights

#Let's create complete flight number (e.g. TG001)
hflights$CF_num <- paste0(hflights$UniqueCarrier,hflights$FlightNum)

#Convert a vector to a single text with collapse
test <- paste0(hflights$UniqueCarrier,hflights$FlightNum, collapse = " ")
length(test)
nchar(test)
```

‘cut()’

- Grouping numerical data into “ordinal” data
- Easier for regression model/scoring system and interpretation

```
140 ####cut####
141 ?women
142 women$weight
143 #Let's group data by weight
144 #-Inf------150]------Inf (i.e Grp 1 = -Inf to 150 and Grp2 = >150 to Inf)
145 cut(women$weight,breaks = c(-Inf,150,Inf) )
146
147 #-Inf---120]---130]---140]---150]---Inf
148 cut(women$weight,breaks = c(-Inf,120,130,140,150,Inf) )
149
150 #-Inf---120)---130)---140)---150)---Inf
151 cut(women$weight,breaks = c(-Inf,120,130,140,150,Inf), right = F )
152
153 #-Inf---120]---130]---140]---150]---Inf
154 #Rename group with labels
155 cut(women$weight,breaks = c(-Inf,120,130,140,150,Inf),
156     labels =  c("very underweight","underweight","normal","overweight","obese"))
```

‘lubridate’ package

- Commands in lubridate package guess date&time data from text

```
####lubridate####
library(lubridate)
today() #Current date
now() #Current date & time

ymd(c("2010-1-1","2010/2/2","2010 3 3")) #Texts are written in Year-Month-Day
mdy(c("Feb 2,2011", "November 21st, 2012")) #Texts are written in Month-Day-Year
dmy_hms(c("11/11/2011 11:11:11", "7 April 2001 1:05:11 PM")) #Texts are written
#in Day-Month-Year-Hour-Minute-Second

dmy(c("12/11/2016","2001-1-11")) #Try and see what happens

a <- dmy_hm("5/6/2014 3:40 PM")#Texts are written in Day-Month-Year-Hour-Minute
b <- dmy_hm("6/6/2014 8:20 AM")

print(a)#Default format of date&time
paste0(day(a),"/",month(a),"/",year(a)," ",hour(a),":",minute(a))#Extract each
#element and re-format date&time
b-a #Simple calculation
difftime(b,a, units = "mins") #Specify units
difftime(b,a, units = "days") #Specify units
```

Practical 4

Practical 4: Merge and tally

1. Merge 'airlines' to 'hflights'
2. Count total flights of these airlines

- HINT: airlines$code matches with hflight$???

```
1  library(hflights)
2  ?hflights
3
4  airlines <- data.frame(code = c("XE","CO","WN","OO","MQ","NW"),
5                         name = c("ExpressJet Airlines, Inc.",
6                                  "Continental Airlines, Inc.",
7                                  "Southwest Airlines Co.",
8                                  "SkyWest  Airlines",
9                                  "Envoy Air Inc.",
10                                 "Northwest Airlines Corp."))
11 print(airlines)
```

Practical 5

Practical 5: Create date data from text

1. Create 'hflights$txt_date' storing texts of complete dates
2. Create 'hflights$date' by converting 'hflights$txt_date' to date data

- HINT 1: paste
- HINT 2: lubridate

```
1  library(hflights)
2  ?hflights
3
4  hflights[,c("Year","Month","DayofMonth")]
```

Practical 6

Practical 6: Create ‘ordinal’ data

1. Classify ‘TaxiOut’ times as follows:
   - 0 to 15 as 'Fast’
   - >15 to 30 as 'Regular’
   - >30 to 60 as 'Slow’
   - >60 as 'Problematic’
2. Tally classified ‘TaxiOut’ times

```
1  library(hflights)
2  ?hflights
3
4  summary(hflights$TaxiOut)
```

Data Transformation

‘aggregate()’

1. Group data by variables
2. Analyze each group with a specified function simultaneously (i.e. faster)

- `aggregate(measure.col ~ grp.col1 + grp.col2, data, FUN, argument_for_function)`

  - measure.col = column to be calculated/analyzed
  - grp.col1, grp.col2,…,grp.coln = columns used for grouping data
  - data = the data frame to be analyzed
  - FUN = function to be used for calculation
  - argument_for_function = additional arguments to be passed to the function

‘aggregate()’

- aggregate(measure.col ~ grp.col1 + grp.col2, data, FUN, argument_for_function)

```
180 ####aggregate####
181 ?iris
182 print(iris)
183
184 #Median of ‘Petal.Length’ by ’Species’
185 median(iris$Petal.Length, na.rm = T)
186 unique(iris$Species)
187 #1) For loop method
188 for(s in unique(iris$Species)){
189   print(s)
190   flags <- which(iris$Species == s)
191   result <- median(iris[flags,"Petal.Length"], na.rm = T)
192   print(result)
193 }
194 #2) Aggregate method
195 aggregate(Petal.Length ~ Species, data = iris, FUN = median, na.rm = T)
```

‘t()’

- Transpose matrix and data frame (2D container)
- Rows <-> Columns
- CAUTION: Data type conversion

```
####t####
?iris
t_iris <- t(iris)

print(iris)
print(t_iris)

summary(iris)
summary(t_iris)
```

‘reshape2’ package: Wide vs Long data format

- Repeated measurement data and time series can be represent in either wide or long format

PatientID	Measurement	BloodSugar
1	1	210
1	2	208
1	3	170
1	4	145
1	5	120
2	1	180
2	2	190
2	3	215
2	4	200
2	5	205

LONG (Row = 1 Measurement)

PatientID	BloodSugar1	BloodSugar2	BloodSugar3	BloodSugar4	BloodSugar5
1	210	208	170	145	120
2	180	190	215	200	205

Wide (Column = 1 Measurement)

‘reshape2’ package: Wide vs Long data format

- R: Data analysis and visualization require “long” formatted data
  - 1 Row = 1 Observation/Measurement
- SPSS and GraphPad require “wide” formatted data
  - Every measurement must be added as a column
- Table for long formatted data usually takes more space
- Table for wide formatted data is easier to read

‘melt()’: Wide -> Long

- Wikipedia PPP data from 1980 to 1989

```
192 ####melt pt 1####
193 #install.packages("rvest", dep = T)
194 library(rvest)
195 library(reshape2)
196 
197 ####For importing data from html source; Do not worry about this####
198 url <- 'https://en.wikipedia.org/wiki/List_of_countries_by_past_and_projected_GDP_(PPP)'
199 test <- read_html(url)
200 tbls <- html_nodes(test, "table")
201 ttt <- html_table(tbls, fill = T)
202 tb4 <- ttt[[4]]
203 #####
204 
205 str(tb4)
206 View(tb4)
```

‘melt()’: Wide -> Long

- Identify your “id.vars” = columns you want to keep

	Country (or dependent territory)	1980	1981	1982	1983	1984	1985	198
1	Afghanistan							
2	Albania	5,547	6,410	7,005	7,362	7,776	7,904	8,5
3	Algeria	86,629	97,559	110,243	120,783	132,073	143,933	146
4	Angola	14,949	15,626	16,595	17,975	19,729	21,073	22,
5	Antigua and Barbuda	267	303	322	352	402	446	508
6	Argentina	176,595	181,994	187,200	201,854	213,199	204,727	223
7	Armenia							
8	Australia	154,525	175,927	186,959	193,419	212,994	231,803	242
9	Austria	84,466	92,261	99,855	106,708	110,861	116,975	122
10	Azerbaijan							
11	Bahamas	1,967	2,089	2,358	2,618	2,776	2,982	3,1
12	Bahrain	6,199	6,967	7,873	8,756	9,446	9,658	9,9
13	Bangladesh	42,099	47,445	52,005	56,549	61,002	65,312	69,
14	Barbados	1,355	1,453	1,467	1,533	1,645	1,716	1,8

‘melt()’: Wide -> Long

- Identify your “id.vars” = columns you want to keep

	Country (or dependent territory)	1980	1981	1982	1983	1984	1985	198
1	Afghanistan							
2	Albania	5,547	6,410	7,005	7,362	7,776	7,904	8,5
3	Algeria	86,629	97,559	110,243	120,783	132,073	143,933	146
4	Angola	14,949	15,626	16,595	17,975	19,729	21,073	22,
5	Antigua and Barbuda	267	303	322	352	402	446	508
6	Argentina	176,595	181,994	187,200	201,854	213,199	204,727	223
7	Armenia							
8	Australia	154,525	175,927	186,959	193,419	212,994	231,803	242
9	Austria	84,466	92,261	99,855	106,708	110,861	116,975	122
10	Azerbaijan							
11	Bahamas	1,967	2,089	2,358	2,618	2,776	2,982	3,1
12	Bahrain	6,199	6,967	7,873	8,756	9,446	9,658	9,9
13	Bangladesh	42,099	47,445	52,005	56,549	61,002	65,312	69,
14	Barbados	1,355	1,453	1,467	1,533	1,645	1,716	1,8

‘melt()’: Wide -> Long

- melt(“name_of_your_data”, id.vars = c(“column_to_keep1”, “column_to_keep2”, …))

```
208 ####melt pt 2####
209 mtb4 <- melt(tb4, id.vars = c('Country (or dependent territory)'))
210 View(mtb4)
```

‘melt()’: Wide -> Long

	Country (or dependent territory)	1980	1981	1982	1983	1984	1985	198
1	Afghanistan							
2	Albania	5,547	6,410	7,005	7,362	7,776	7,904	8,5
3	Algeria	86,629	97,559	110,243	120,783	132,073	143,933	146
4	Angola	14,949	15,626	16,595	17,975	19,729	21,073	22,
5	Antigua and Barbuda	267	303	322	352	402	446	508
6	Argentina	176,595	181,994	187,200	201,854	213,199	204,727	223
7	Armenia							
8	Australia	154,525	175,927	186,959	193,419	212,994	231,803	242
9	Austria	84,466	92,261	99,855	106,708	110,861	116,975	122
10	Azerbaijan							
11	Bahamas	1,967	2,089	2,358	2,618	2,776	2,982	3,1
12	Bahrain	6,199	6,967	7,873	8,756	9,446	9,658	9,9
13	Bangladesh	42,099	47,445	52,005	56,549	61,002	65,312	69,
14	Barbados	1,355	1,453	1,467	1,533	1,645	1,716	1,8

	Country (or dependent territory)	variable	value
1	Afghanistan	1980	
2	Albania	1980	5,547
3	Algeria	1980	86,629
4	Angola	1980	14,949
5	Antigua and Barbuda	1980	267
6	Argentina	1980	176,595
7	Armenia	1980	
8	Australia	1980	154,525
9	Austria	1980	84,466
10	Azerbaijan	1980	
11	Bahamas	1980	1,967
12	Bahrain	1980	6,199
13	Bangladesh	1980	42,099
14	Barbados	1980	1,355
15	Belarus	1980	

‘dcast()’:Long -> Wide

- ‘ChickWeight’ data
- Identify 3 types of columns
  1. Columns to be kept (similar to id.vars of ‘melt())
  2. Columns to be converted to column names (usually columns showing time of measurement)
  3. **<u>A column</u>** storing values to be shown in each cell of the ‘dcasted’ table

	weight	time	chick	diet
1	42	0	1	1
2	51	2	1	1
3	59	4	1	1
4	64	6	1	1
5	76	8	1	1
6	93	10	1	1
7	106	12	1	1
8	125	14	1	1
9	149	16	1	1
10	171	18	1	1
11	199	20	1	1
12	205	21	1	1
13	40	0	2	1

‘dcast()’:Long -> Wide

- ‘ChickWeight’ data
- Identify 3 types of columns
  1. Columns to be kept (similar to id.vars of ‘melt())
  2. Column to be converted to column names (usually column showing time of measurement)
  3. A column storing values to be shown in each cell of the ‘dcasted’ table

	3	2	1	
	weight	time	chick	diet
1	42	0	1	1
2	51	2	1	1
3	59	4	1	1
4	64	6	1	1
5	76	8	1	1
6	93	10	1	1
7	106	12	1	1
8	125	14	1	1
9	149	16	1	1
10	171	18	1	1
11	199	20	1	1
12	205	21	1	1
13	40	0	2	1

‘dcast()’:Long -> Wide

- ‘ChickWeight’ data
- dcast(“data name”, “Columns to be kept1+Columns to be kept2+…” ~ “timing column1 + timing column2”, value.vars = column storing values)
- TYPOS in the demo R-script!

```
221 ####dcast pt####
222 library(reshape2)
223
224 View(ChickWeight)
225
226 cast_CW <- dcast(ChickWeight, Chick + Diet ~ Time, value.var = "weight")
227
228 View(cast_CW)
```

‘dcast()’:Long -> Wide

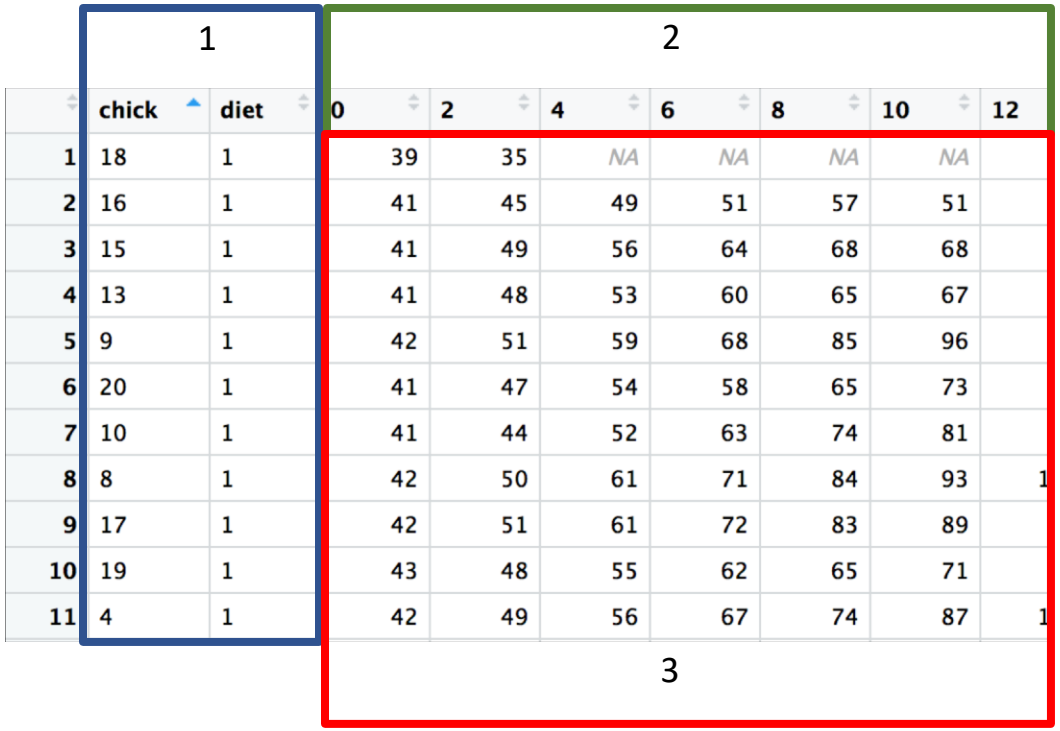
- ‘ChickWeight’ data

	chick	diet	0	2	4	6	8	10	12
1	18	1	39	35	NA	NA	NA	NA	
2	16	1	41	45	49	51	57	51	
3	15	1	41	49	56	64	68	68	
4	13	1	41	48	53	60	65	67	
5	9	1	42	51	59	68	85	96	
6	20	1	41	47	54	58	65	73	
7	10	1	41	44	52	63	74	81	
8	8	1	42	50	61	71	84	93	1
9	17	1	42	51	61	72	83	89	
10	19	1	43	48	55	62	65	71	
11	4	1	42	49	56	67	74	87	1

‘dcast()’:Long -> Wide

- ‘ChickWeight’ data

	3	2	1	
	weight	time	chick	diet
1	42	0	1	1
2	51	2	1	1
3	59	4	1	1
4	64	6	1	1
5	76	8	1	1
6	93	10	1	1
7	106	12	1	1
8	125	14	1	1
9	149	16	1	1
10	171	18	1	1
11	199	20	1	1
12	205	21	1	1
13	40	0	2	1



	1		2						
	chick	diet	0	2	4	6	8	10	12
1	18	1	39	35	NA	NA	NA	NA	
2	16	1	41	45	49	51	57	51	
3	15	1	41	49	56	64	68	68	
4	13	1	41	48	53	60	65	67	
5	9	1	42	51	59	68	85	96	
6	20	1	41	47	54	58	65	73	
7	10	1	41	44	52	63	74	81	
8	8	1	42	50	61	71	84	93	1
9	17	1	42	51	61	72	83	89	
10	19	1	43	48	55	62	65	71	
11	4	1	42	49	56	67	74	87	1

3

Practical 7

Practical 7: Melt and Aggregate

1. Melt table 'tb’
2. Correct data type of each column
3. Use 'aggregate()' to find mean PPP from 2000 to 2009 for each country

```r
library(rvest)
library(reshape2)

####For importing data from html source; Do not worry about this####
url <- 'https://en.wikipedia.org/wiki/List_of_countries_by_past_and_projected_GDP_(PPP)'
test <- read_html(url)
tbls <- html_nodes(test, "table")
ttt <- html_table(tbls, fill = T)
tb <- ttt[[10]]
colnames(tb)[1] <- "country" #Change the name of the first column
for(i in 2:ncol(tb)){
  tb[,i] <- gsub(pattern = ",",replacement = "",x = tb[,i]) #Remove ',' from numbers
}
```

Practical 8

Practical 8: dcast

- Use 'dcast' to create a wide format table for 'sleepstudy' data

```
1  #install.packages("lme4",dep = T)
2  library(lme4)
3  library(reshape2)
4
5  ?sleepstudy
6  View(sleepstudy)
7
```