# Online Academic Data Analysis Bootcamp Using Open-Access Program R

## Session 2

# Getting data into and out of R


**Dr. Patrick Njage**

**Patrick Njage (BSc, MSc, Mstat, PhD)**

**Technical University of Denmark**

**Academic Data Analysts** (WWW.ACADEMICDATAANALYSTS.ORG/)

**Institute of Statistics and Data Science (ISADS)**

(WWW.ISADS.ORG )

# Outline

**Creating variables and data frames**

**Calculating new variables from existing ones**

**Working with dates**

**Missing values**

**Entering data, creation of variables and coding variables using R Commander**

**Using other software to enter and edit data**

**Importing data**

**Exporting data**

**Viewing and Examining data**

# Getting data into R: Creating variables

- Previously in Lecture 1, the c() function was to create objects that contain data.

```
studentNames<-c("John","James","Albert","Ahmed", "Aarya")

studentGrades1 <- c(60, 87, 53, 72, 85) #Grades at test 1
```

- Note: names (text data is referred to as strings) are in quotes, but entered the ages (numbers referred to as numeric) are not.
- R will automatically treat studentNames as nominal because it is a string variable

# Getting data into R:  Creating dataframes

- We have two separate objects: *studentNames* and *studentGrades1*
- More efficient to combine the files into a single object: dataframe (like spreadsheets in excel etc)
- Use the data.frame() function while renaming *studentNames* to Name and *studentGrades1* to First_Grade:

students <-data.frame(Name = *studentNames*, First_Grade = *studentGrades1*)

- Run below to see the dataframe:

students

# Getting data into R: Creating dataframes

dataframe$variableName can be used to refer to the variables

e.g. to get student Name data run students$Name, similarly, for First Grade variable we could use student$First_Grade

- We can add another variable e.g. Second Grades using the c() function as follows:

students$Second_Grade <-c(65, 56, 87, 79, 85) #run students to view the new dataframe

- You can view variable names using:

names(students)

# Getting data into R: Calculating new variables from existing ones

- Use of basic operators in R

```
students$Improvement <- students$Second_Grade – students$First_Grade
```

This inserts a new variable “Improvement” to the students dataframe.

Note: always advised to combine separate words in variable names e.g. First_Grade or FirstGrade or First.Grade not First Grade

# Getting data into R: Creating a date variable

- Dates: same procedure as with a string variable, but particular format
- Need to tell R that the data are dates to enable date-related computations e.g. calculate age at an event by taking the events date and subtract from it the date they were born.
- Text converted into date objects using the as.Date() function.
- This function takes strings of text, and converts them into dates

# Getting data into R: Creating a date variable

- Examples: calculate age difference using birthdates from four couples
- Use the as.Date() function:

```
husband<-as.Date(c("1973-06-21", "1970-07-16", "1949-10-08", "1969-05-24"))
wife<-as.Date(c("1984-11-12", "1973-08-02", "1948-11-11", "1983-07-23"))
```

- Calculate the difference between the two variables:

```
agegap <- husband-wife
agegap
```

- Time differences in days

```
[1] -4162 -1113   331 -5173
```

# Creating coding variables/factors

- Numbers represent different groups of data i.e. numbers represent names/ nominal variables

  E.g. levels of a treatment variable in an experiment, different groups of people (men or women, an experimental group or a control group, ethnic groups, etc.), different geographic locations, different organizations, etc.

- R Functions for factors:

*factor(x = character(), levels, labels = levels)*

*newFactor<-gl(number of levels, cases in each level, total cases, labels =c("label1", "label2"…))*

# Creating coding variables/factors

- Example:  Data where 1 = lecturers and 2 = students therefore labels of “Lecturer” and “Student”
- Write levels = c(“Lecturers”, “Students”).

Code

job<-c(1,1,1,1,1,2,2,2,2,2) or better use rep (number of repetitions) job<-c(rep(1, 5),rep(2, 5))

job<-factor(job, levels = c(1:2), labels = c("Lecturer", "Student"))

Or

job<-gl(2, 5, labels = c("Lecturer", "Student"))

- Check factor levels

levels(job)

- Rename factor levels e.g. to *Medical Lecturer* and *Medical Student*

levels(job)<-c("Medical Lecturer", "Medical Student")

# Value Labels: ordinal data

```
mydata$v1 <- ordered(mydata$y,
    levels = c(1,3, 5),
    labels = c("Low", "Medium", "High"))
```

Use the **factor()** function for **nominal data** and the **ordered()** function for **ordinal data**. **R** statistical and graphic functions will then treat the data appropriately.

Note: factor and ordered are used the same way, with the same arguments. The former creates factors and the later creates ordered factors.

# Missing Data

In **R**, missing values are represented by the symbol **NA** (not available) . Impossible values (e.g., dividing by zero) are represented by the symbol **NaN** (not a number). Unlike SAS, **R** uses the same symbol for character and numeric data.

**Testing for Missing Values**

```
is.na(x) # returns TRUE of x is missing
y <- c(1,2,3,NA)
is.na(y) # returns a vector (F F F T)
```

# Missing Data

**Recoding Values to Missing**

```
# data frame that codes missing values as 99
df <- data.frame(variable1 = c(1:3, 99), variable2 = c(2.5, 4.2, 99, 3.2))
# change 99s to NAs
df[df == 99] <- NA
```

**Excluding Missing Values from Analyses**

Arithmetic functions on missing values yield missing values.

```
x <- c(1,2,NA,3)
mean(x)          # returns NA
mean(x, na.rm=TRUE) # returns 2
```

# Missing Data

The function **complete.cases()** returns a logical vector indicating which cases are complete.

```
# list rows of data that have missing values
mydata[!complete.cases(mydata),]
```

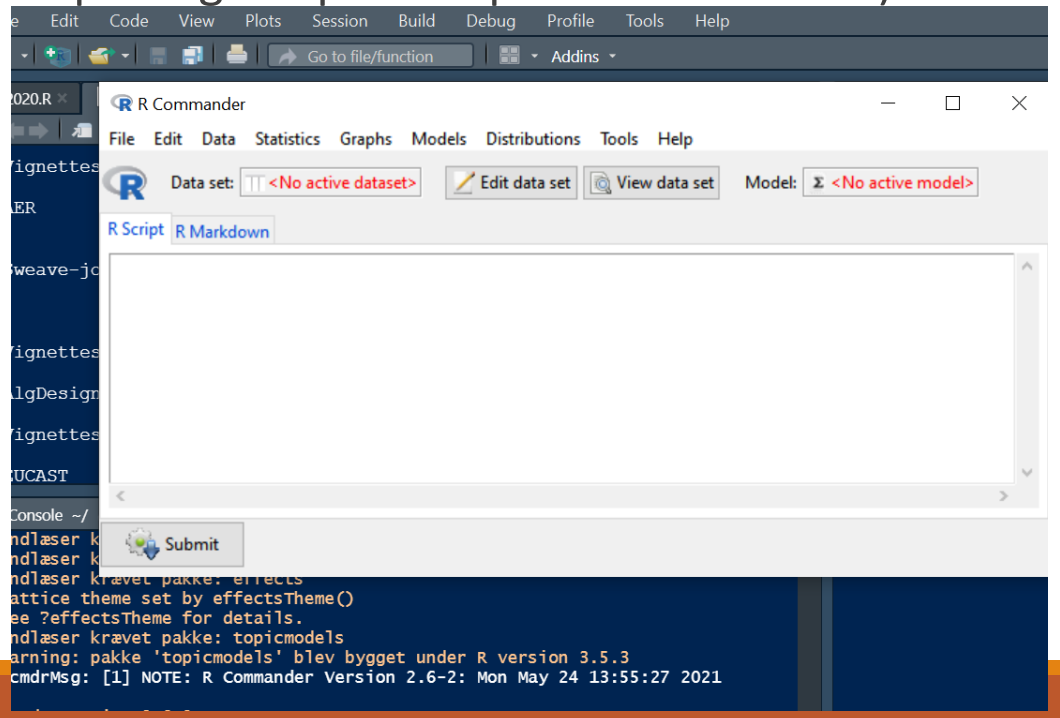
The function **na.omit()** returns the object with listwise deletion of missing values.

```
# create new dataset without missing data
newdata <- na.omit(mydata)
```

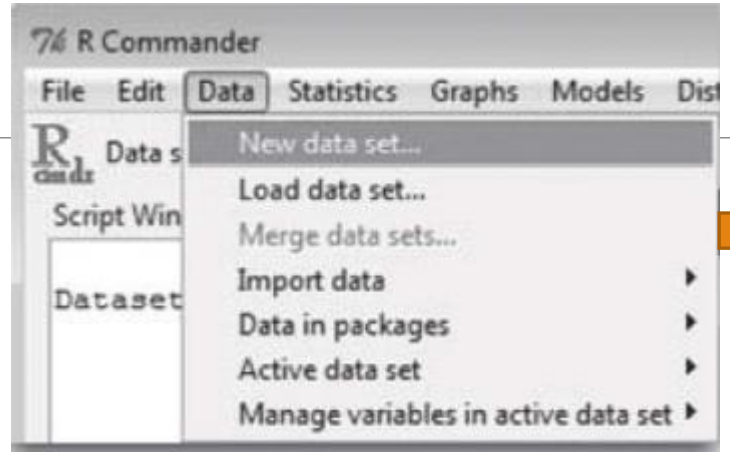
# Entering data with R Commander

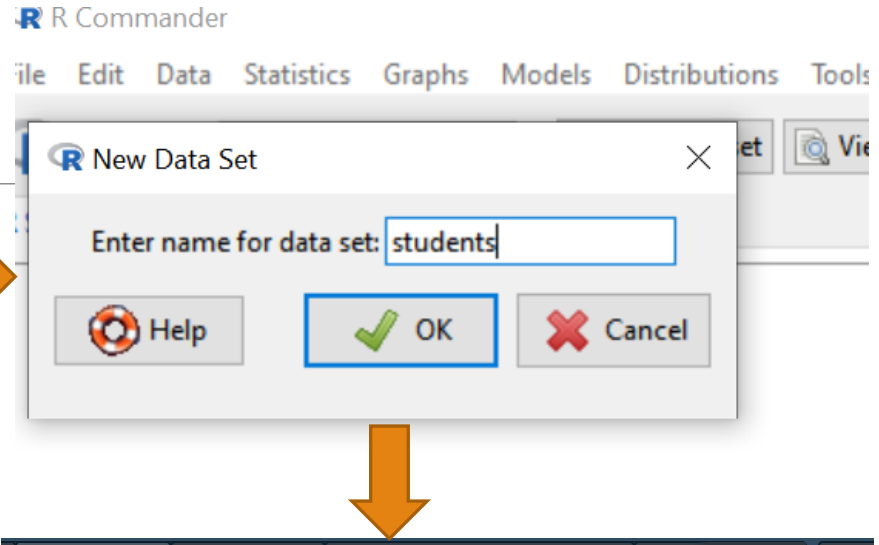
- Package Rcmdr supports basic data editing (and analysis) using a windows style interface for basic data manipulation and analysis.
- Install and load Rcmdr (Note use of dependencies = TRUE- when a package uses other packages, these are known as dependencies (because the package depends upon them to work)

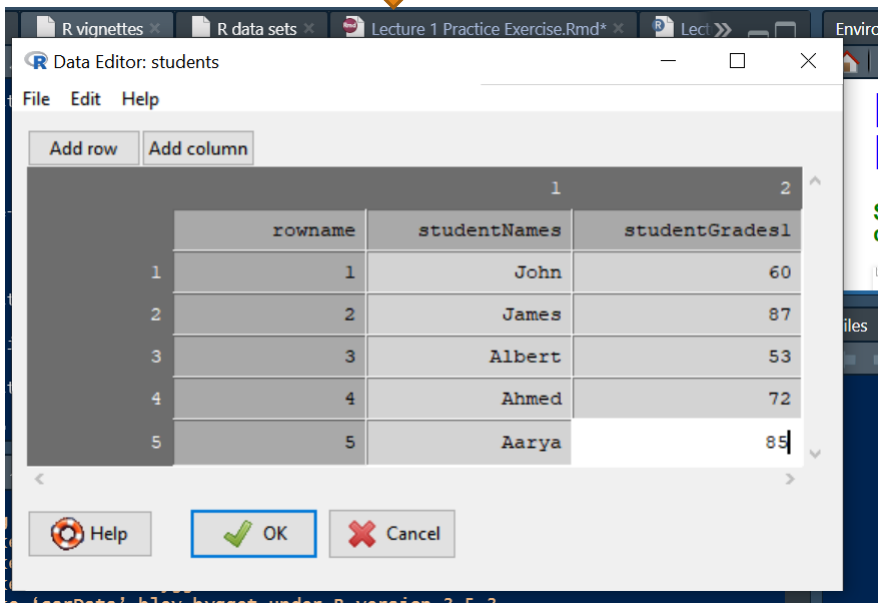
```
install.packages("Rcmdr", dependencies = TRUE)
library(Rcmdr)
```



# Creating variables and entering data with R Commander



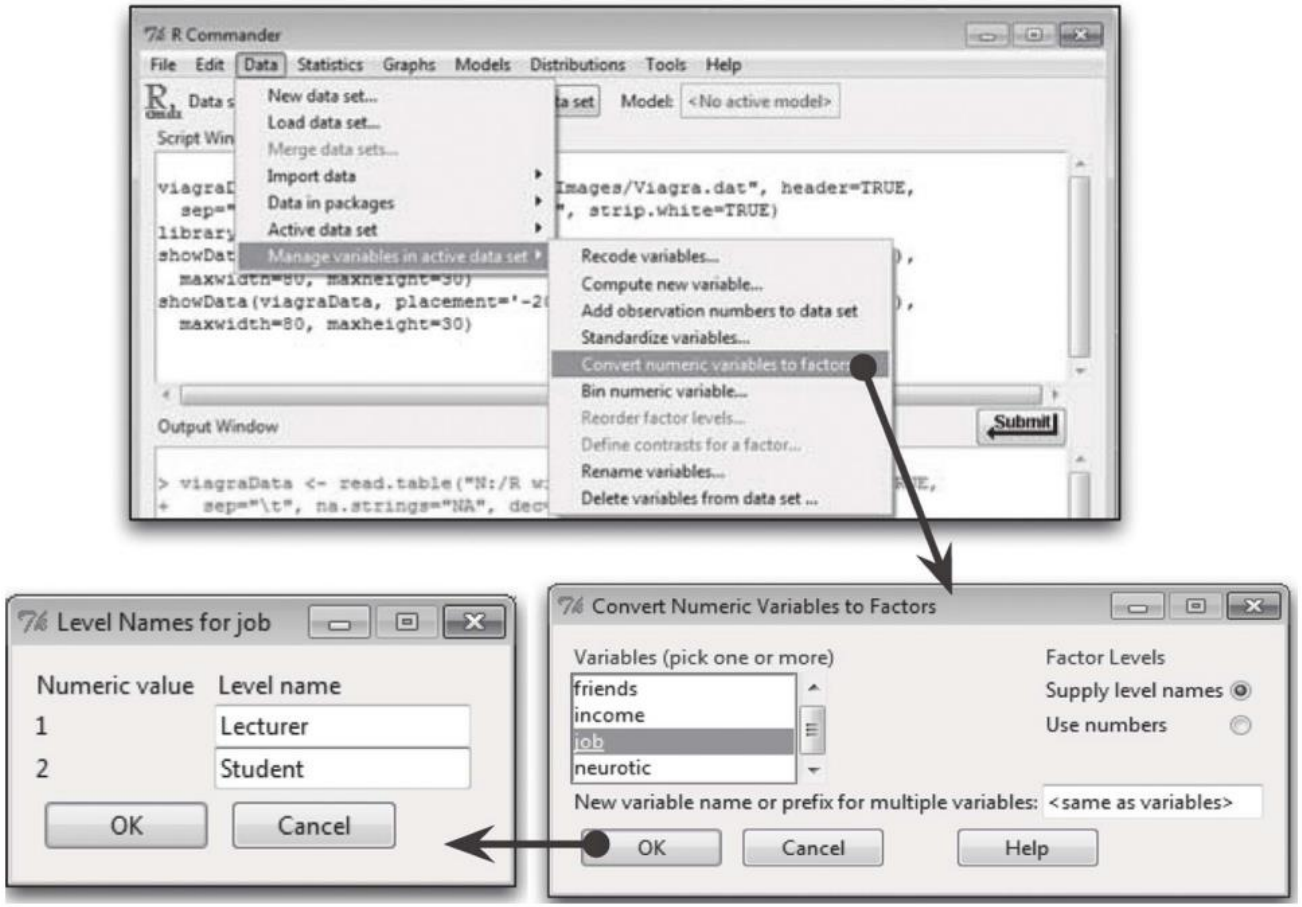

Data⇒New data set…, (opens a dialog box that enables you to name the dataframe).

Enter data

To save the data, simply close this window.

# Creating coding variables with R Commander

R Commander by selecting the Data⇒Manage variables in active data set⇒Convert numeric variables to factors



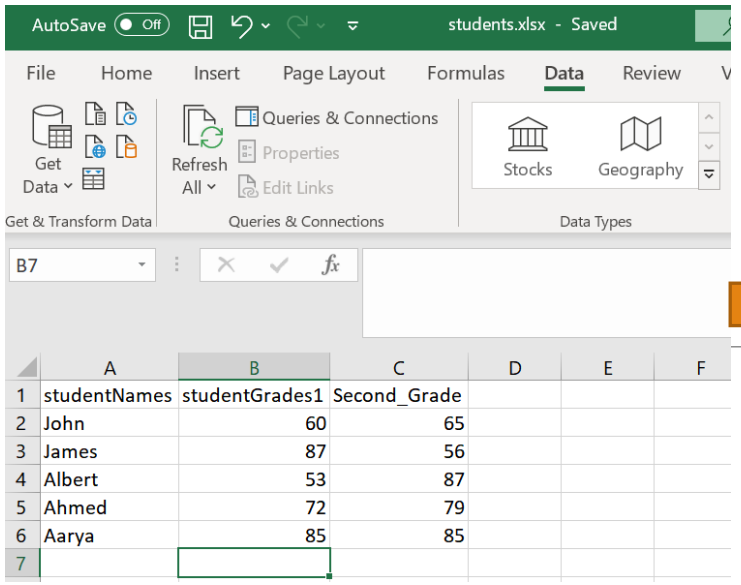
# Importing data: formats

- For large datasets, data entry is usually via other software that with a spreadsheet style window
- Microsoft Excel most widely used as well as specialist packages such as SPSS and SAS
- Common data entry:  each person/ case represented by a row in the spreadsheet, whereas each variable is represented as a column
- Two most commonly used R-friendly formats are tab-delimited text (.txt in Excel and .dat in SPSS) and comma-separated values (.csv)
- Both are plain text files very easy to export from Excel and other software packages
- Next two slides show the process

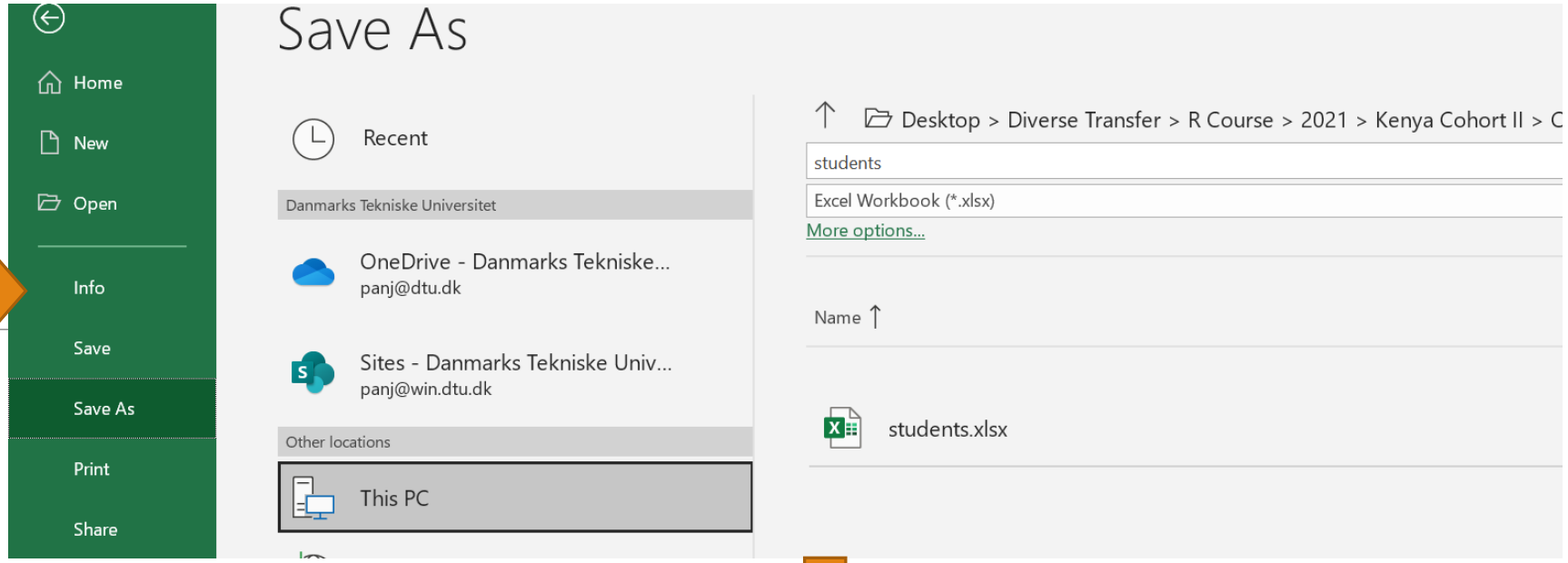
# Importing data: formats- excel

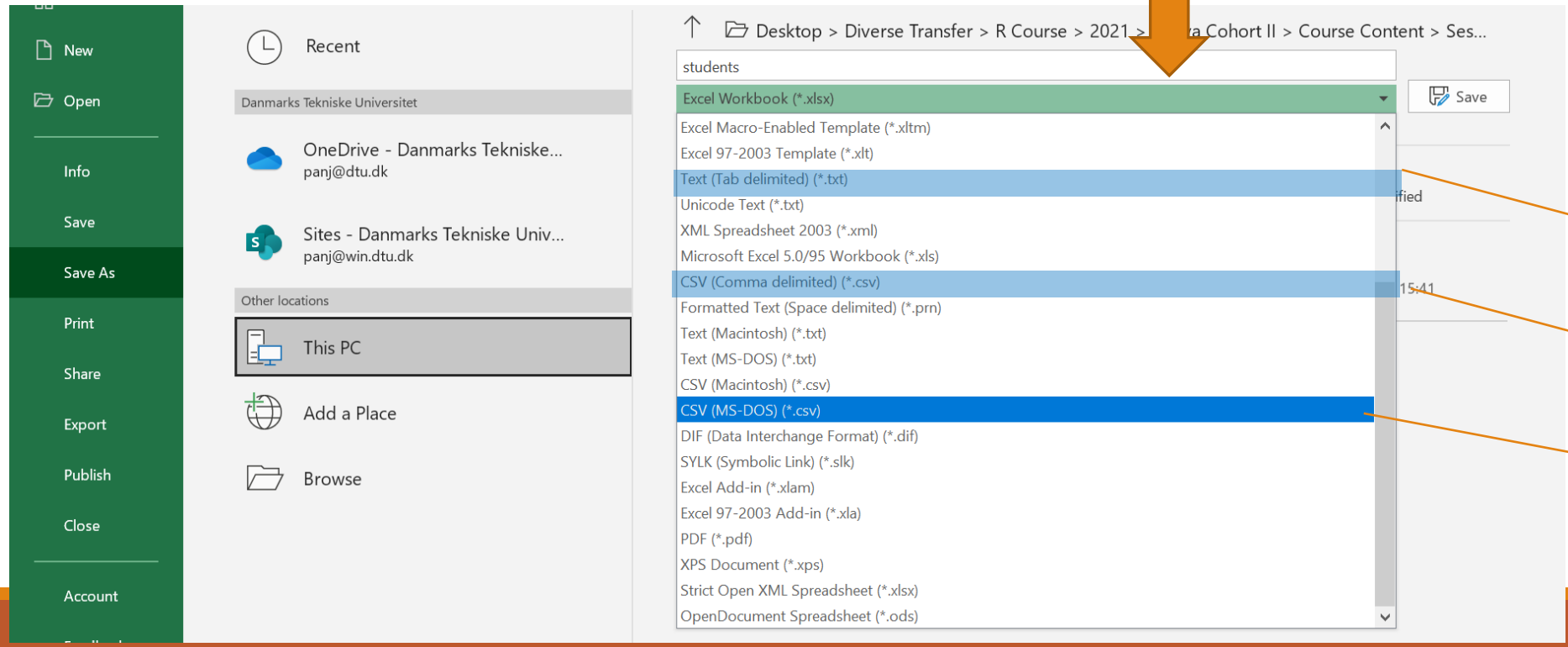
- Excel: after data entry, open the Save As... dialog box
- Select the location you wish to save the file (esp. R Working Directory)
- Change Excel's default format (.xlsx or .xls)  by clicking on the drop-down list


- A variety of file types are available - two best for R are Text (Tab delimited) and CSV (Comma delimited)
- Select one of these file types, type a name for your file and click "Save"
- The saved file is either a .txt file or a .csv  (look at the file extension)
- Similar process for exporting data from SPSS (and other packages)




| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | studentNames | studentGrades1 | Second_Grade | | | |
| 2 | John | 60 | 65 | | | |
| 3 | James | 87 | 56 | | | |
| 4 | Albert | 53 | 87 | | | |
| 5 | Ahmed | 72 | 79 | | | |
| 6 | Aarya | 85 | 85 | | | |
| 7 | | | | | | |




Save as either:

- Text (Tab Delimited)
- CSV (Comma delimited)
- CSV (MS DOS

# Importing data: data saved in working directory

**Import data from CSV**

This is the code we use to import csv file into R

```
GSSsubset_df <- read.table("GSSsubset.csv", header=TRUE, sep=",") # Or this code:
GSSsubset_df <- read.csv("GSSsubset.csv",header=T,as.is=T)
```

**Import data from TXT**

```
GSSsubset_df <- read.delim("GSSsubset.txt", as.is=TRUE, header=T) # Or this code:
GSSsubset_df <- read.delim("GSSsubset.txt", header=T, strings=F)
```

**Import data from Excel**

```
library(readxl)
DataNameinR <- read_excel("ExcelFilename.xlsx",     sheet = “ExcelSheetName")
GSSsubset <- read_excel("GSSsubset.xlsx", sheet = "Sheet1")
```

# Import dataset: from folder with pathway

- read.csv("full path")
  - Example:

```
gss <- read.csv("C:/Users/Desktop/R Introduction workshop/GSSsubset.csv")
```

R uses Forward Slash "/" in the directory path: C:/Users/R workshop

- For mac users, you can just copy and paste the path.
- For Windows users, you need to change all the back slash to forward slash

# Import dataset: From A Comma Delimited Text File (.CSV)

```
# first row contains variable names, comma is separator
# assign the variable id to row names
# note the / instead of \ on MS Windows systems

mydata <- read.table("c:/mydata.csv", header=TRUE,
sep=",", row.names="id")
```

# Importing data from other packages: SPSS

**Import data from SPSS**

R can import datasets from SPSS with the function read.spss() from the package foreign. Alternatively, the function spss.get() from Hmisc package can be used. While foreign is a default package in R, the Hmisc package need to be installed.

Here is an example:

```
library(foreign)

Sleep_df <- read.spss("sleep.sav", use.value.label=TRUE,
to.data.frame=TRUE)
```

# Importing data from other packages

**Import data from Stata**

To import a dataset from Stata into R, the function read.dta() from foreign package is used. More specifically look the code below:

```
library(foreign)

census_df <- read.dta("census.dta")
```

census_df is the name of data frame in R, and census.dta is the file name of Stata dataset we want to import.

**Import data from SAS**

```
install.packages("haven")

library(haven)

airline_df = read_sas("airline.sas7bdat”)
```

# Exporting Data

There are numerous methods for exporting **R** objects into other formats .

For SPSS, SAS and Stata we will demonstrate use of with **foreign** package.

For Excel, we will use the **writexl** package.

# Exporting Data

**To A Tab Delimited Text File**

```
write.table(GSSsubset, “GSSExported.txt", sep="\t")
```

**To an Excel Spreadsheet**

```
install.packages("writexl")
library("writexl")
write_xlsx(GSSsubset, "GSSExported.xlsx")
```

**To SAS**

```
library(foreign)
  write.foreign(mydata, "c:/mydata.txt", "c:/mydata.sas",   package="SAS")
```

# Viewing Data

**There are a number of functions for listing the contents of an object or dataset.**

```
# list objects in the working environment
    ls(GSSsubset)
# list the variables in mydata
    names(GSSsubset)
# list the structure of mydata
    str(GSSsubset)
# list levels of factor v1 in mydata
    levels(GSSsubset$degree)
# dimensions of an object
    dim(GSSsubset)
```

# Viewing Data

**A number of functions for listing the contents of an object or dataset.**

```
# class of an object (numeric, matrix, dataframe, etc)
class(GSSsubset)

# print mydata
GSSsubset

# print first 10 rows of mydata
head(GSSsubset, n=10)

# print last 5 rows of mydata
tail(GSSsubset, n=5)
```

# Examine dataset

dim(GSSsubset) # how many rows and columns

nrow(GSSsubset) #how many rows

ncol(GSSsubset) #how many columns

colnames(GSSsubset)  # what are the names of columns- like the function names(GSSsubset)

rownames(GSSsubset)

# R ”tips and tricks”

- Specifying file locations in R may be new for those used to selecting files through dialog boxes rather than typing file extensions
- Setting your working directory if a very efficient solution
- An alternative is to use the *file.choose()* function.
- **This function opens a standard dialog box allowing you to navigate to the file you want.**
- You can incorporate this function into read.csv() and read.delim() as follows:

```
gssData<-read.csv(file.choose(), header = TRUE)

gssData<-read.delim(file.choose(), header = TRUE)
```

# Tips: Best practices on saving Excel datasets for import into R

Make sure that your data is well prepared to be imported - neglecting this may lead to errors in importation

- The first row of the spreadsheet is usually reserved for the header, while the first column is used to identify the sampling unit;
- Avoid names, values or fields with blank spaces. Otherwise, each word will be interpreted as a separate variable, resulting in errors that are related to the number of elements per line in your data set;

# Tips: Best practices on saving Excel datasets for import into R

- If you would If you want to concatenate words, do this by inserting a . Or _.For example:

  Sepal.Length or Sepal_Length

- Short names are preferred over longer names;
- Avoid using names that contain symbols such as ?, $,%, ^, &, *, (, ),-,#, ?,,,<,>, /, |, \, [ ,] ,{, and };
- Delete any comments that you have made in your Excel file to avoid extra columns or NA's getting added to your file; and
- Make sure that any missing values in your data set are indicated with NA.

# Thank You