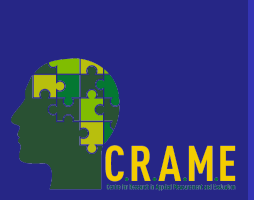


Okan Bulut ©
bulut@ualberta.ca

R you ready to learn R?

Okan Bulut

Centre for Research in Applied Measurement and Evaluation (CRAME)

March 6, 2018

(http://bit.ly/rworkshop2018)

Workshop Details

Okan Bulut ©
bulut@ualberta.ca

- This workshop is intended for graduate students, faculty members, and researchers who want to learn about statistical analysis and data visualization using **R**.
- My contact information:
  - Dr. Okan Bulut
  - Measurement, Evaluation, and Data Science
  - Department of Educational Psychology
  - Email: bulut@ualberta.ca
  - Website: https://sites.ualberta.ca/~bulut/
- Workshop materials can be downloaded from:
  - http://bit.ly/rworkshop2018

Goals of This Workshop

1. To provide a brief introduction to R for beginners
2. To demonstrate how to use **R** and **RStudio** for:
   - describing and summarizing data
   - data visualization
   - statistical data analysis

Outline

Okan Bulut ©
bulut@ualberta.ca

1. Part I: An overview of **R** and **RStudio**
   - What is **R**?
   - What is **RStudio**?
   - Basics of the **R** language
   - Data entry and management
2. Break
3. Part II: Descriptive and inferential statistics with **R**
   - Descriptive statistics
   - Visualizing data in **R**
   - Inferential statistics (e.g., t tests, correlation, regression)

Statistical Software Programs

- General purpose programs ($-$$):
  - **R**
  - SPSS
  - SAS
  - STATA
  - Minitab
  - Microsoft Excel (partially)
- Specialized programs ($-$$$):
  - Mx
  - AMOS
  - Mplus
  - LISREL
  - Your favorite program

What is R?

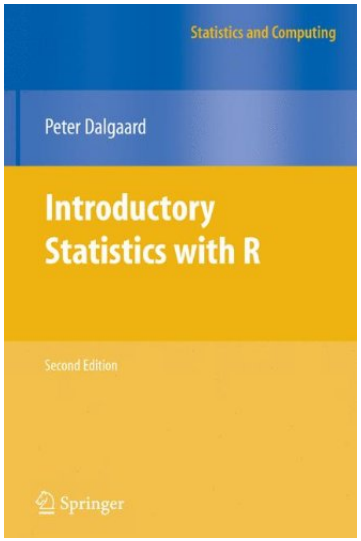
R:

- is a free, open source program for statistical computing and data visualization.
- is cross-platform (e.g., available on Windows, Mac OS, and Linux).
- is maintained and reguarly updated by the Comprehensive R Archive Network (CRAN; https://cran.r-project.org/).
- is capable of running all types of statistical analyses.
- has amazing visualization capabilities (high-quality, customizable figures).
- enables reproducible research.
- has many other capablities, such as web programming.
- supports user-created packages (currently, more than 10,000)

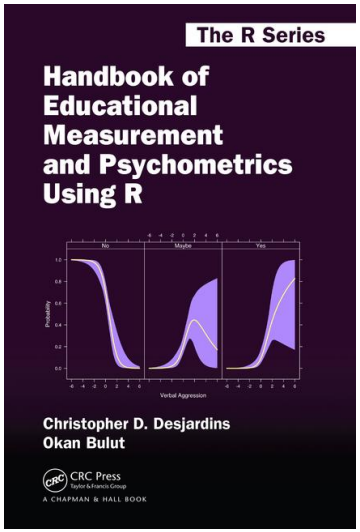
Some R Resources

There are many websites and free e-books about **R** available on the Internet. A short list of some **R** resources are below:

- R User Manuals: https://cran.r-project.org/manuals.html
- Quick R: http://www.statmethods.net/index.html
- R Cookbook: http://www.cookbook-r.com/
- Advanved R: http://adv-r.had.co.nz/
- Using R for Introductory Statistics: https://goo.gl/owJbLg



https://goo.gl/zt7wc7



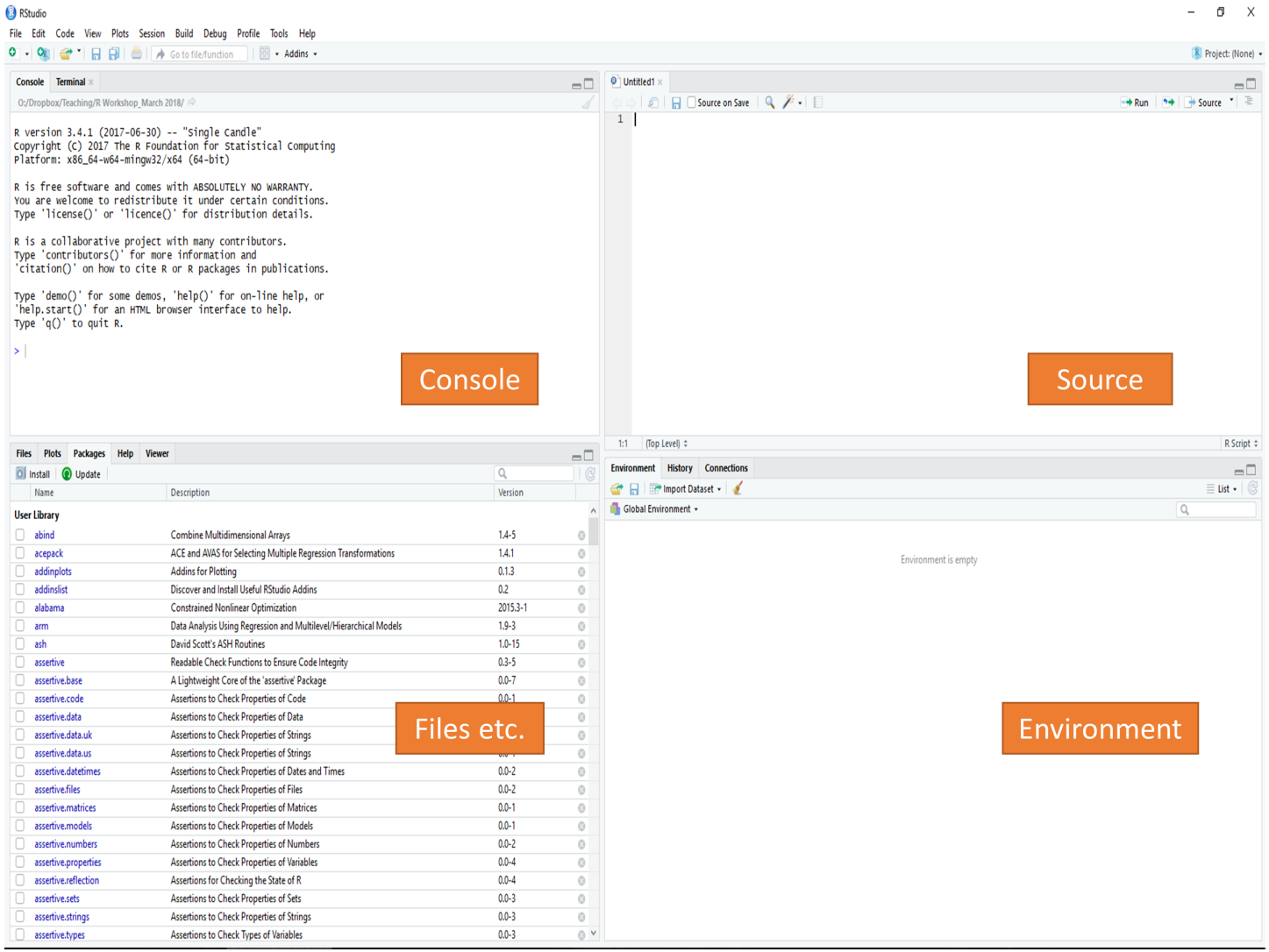
https://goo.gl/Y6X3Hq

What is RStudio?

- **RStudio** is a free program available from the RStudio website.
- **RStudio** provides a more user-friendly interface for **R**.
- **RStudio** includes a set of tools to help you be more productive with **R**, such as:
  - A syntax-highlighting editor for highlighting your **R** codes
  - Functions for helping you type the **R** codes (auto-completion)
  - A variety of tools for creating and saving various plots (e.g., histograms, scatterplot)
  - A workspace management tool for importing or exporting data
- To benefit from **RStudio**, both **R** and **RStudio** should be installed in your computer.

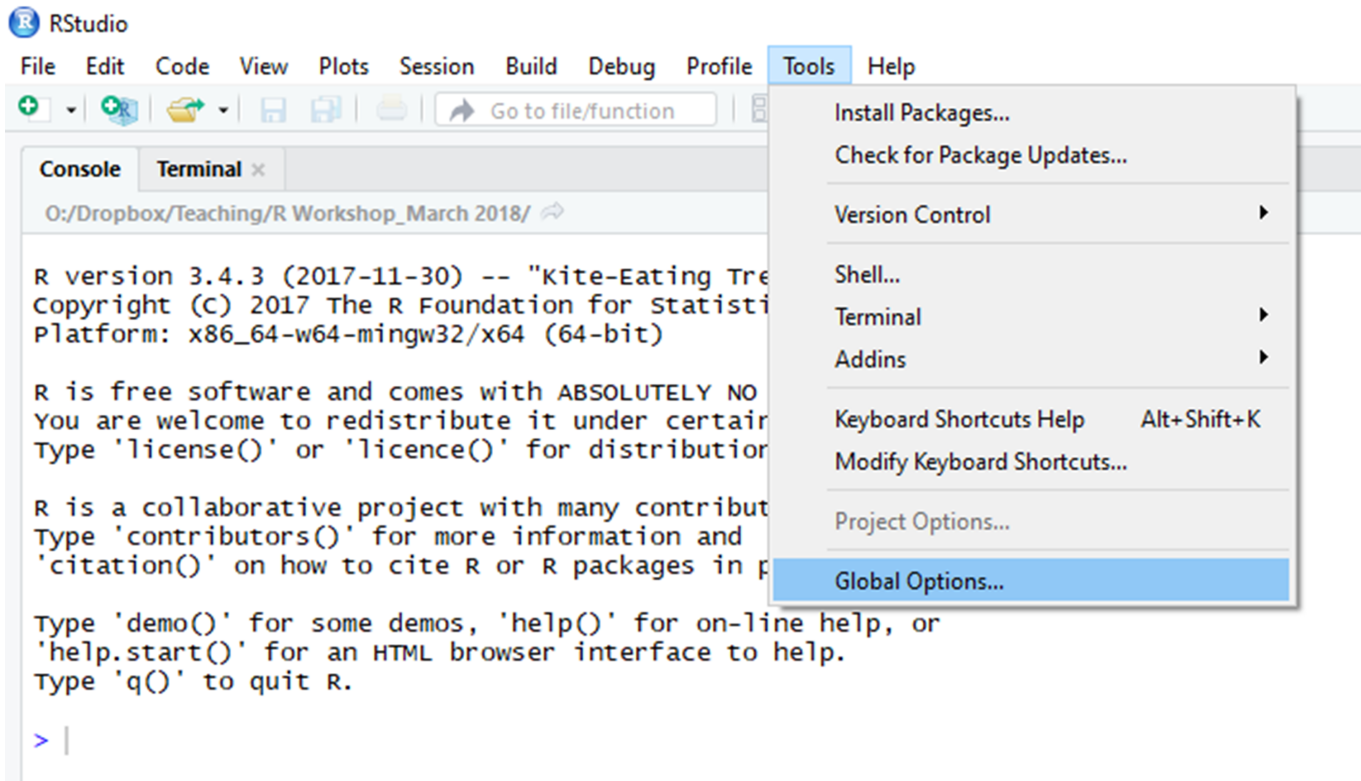
Preview of RStudio

Okan Bulut ©
bulut@ualberta.ca

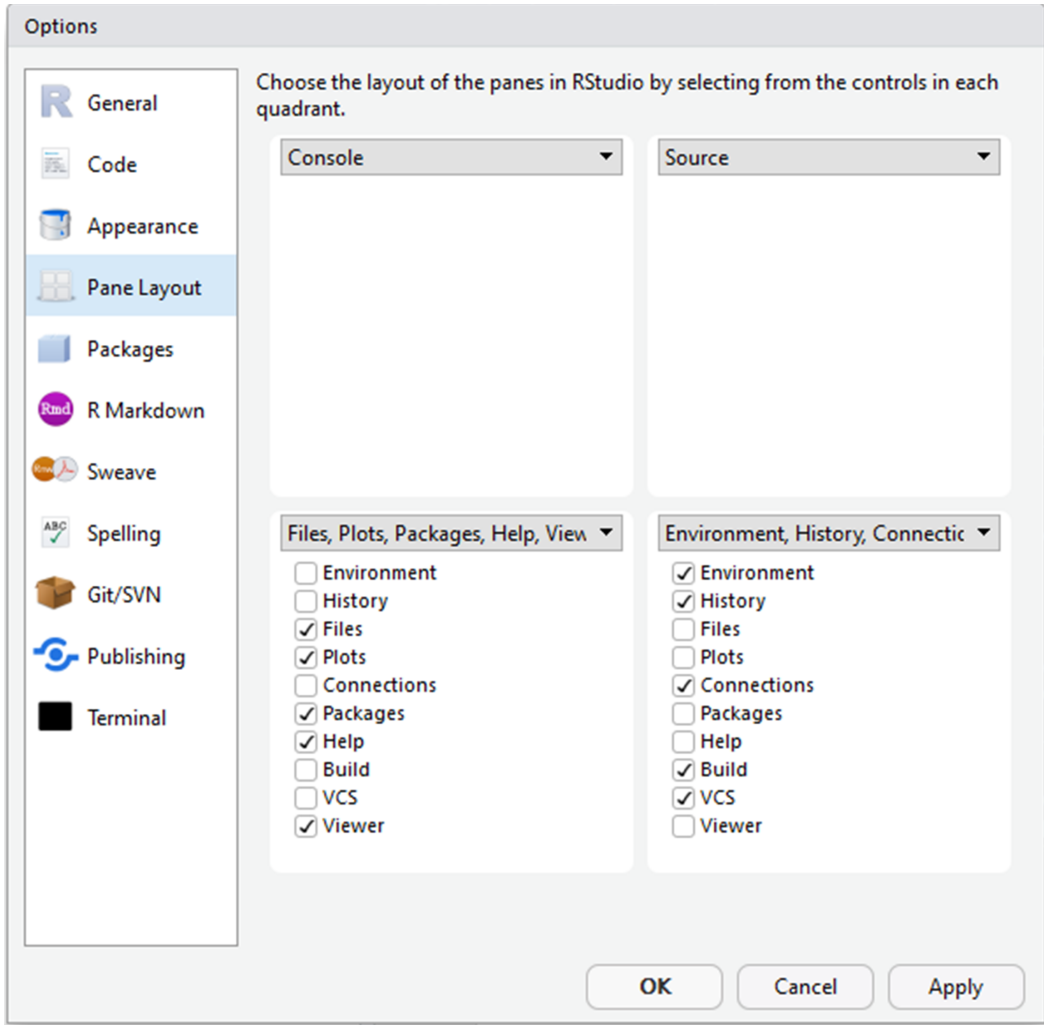


Changing the Pane Layout

The pane layout can be updated using "Global Options". I personally prefer console on the top-left, source on the top-right, files on the bottom-left, and environment on the bottom-right.

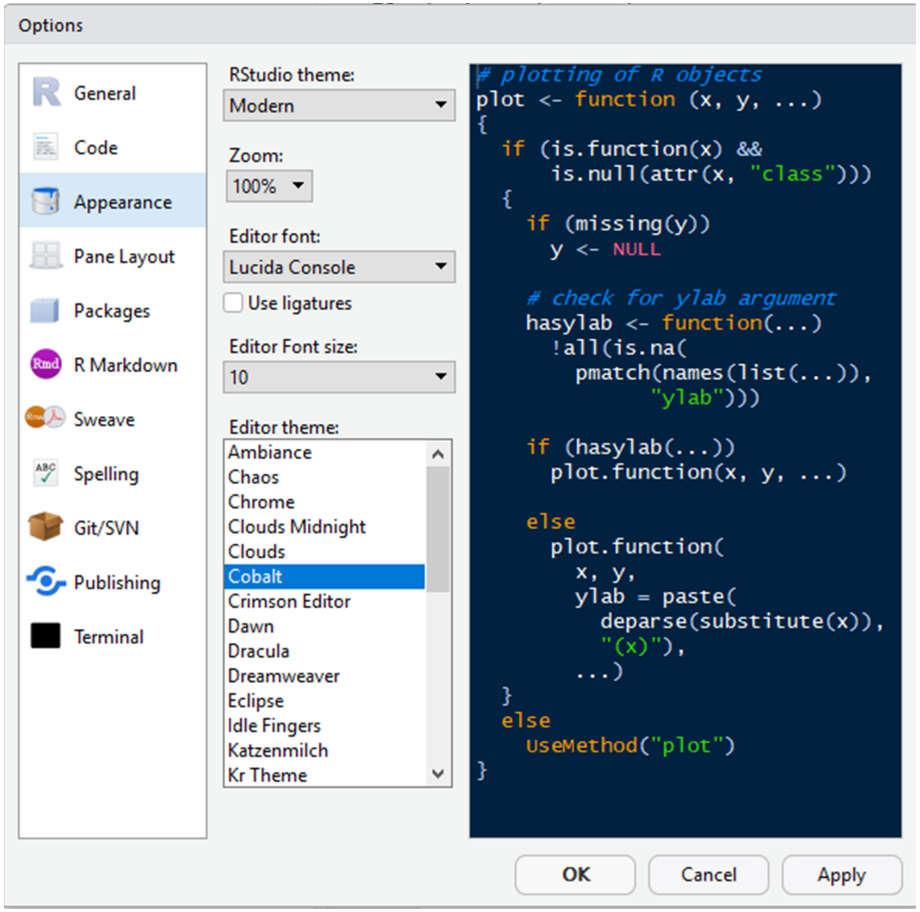


Changing the Pane Layout



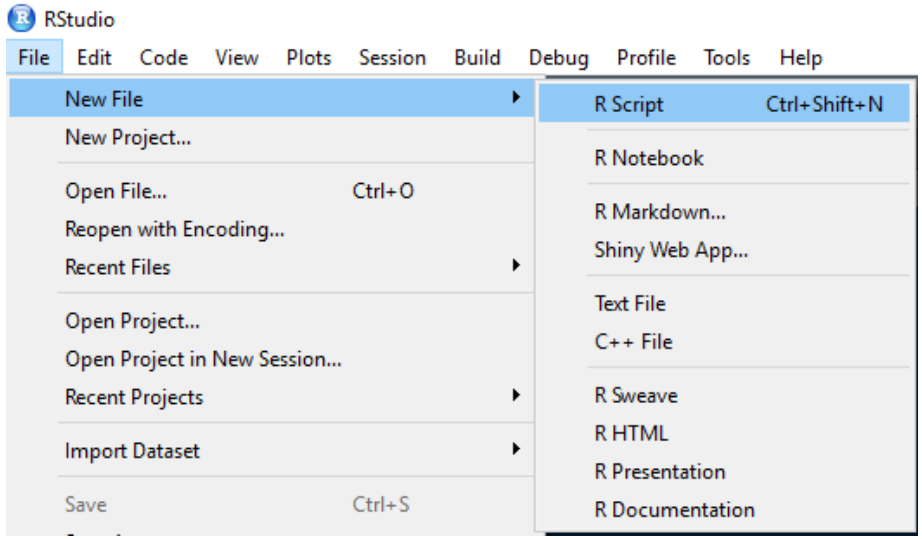
Changing the Appearance Settings

We can also change the appearance (e.g., code highlighting, font type, font size, etc.)



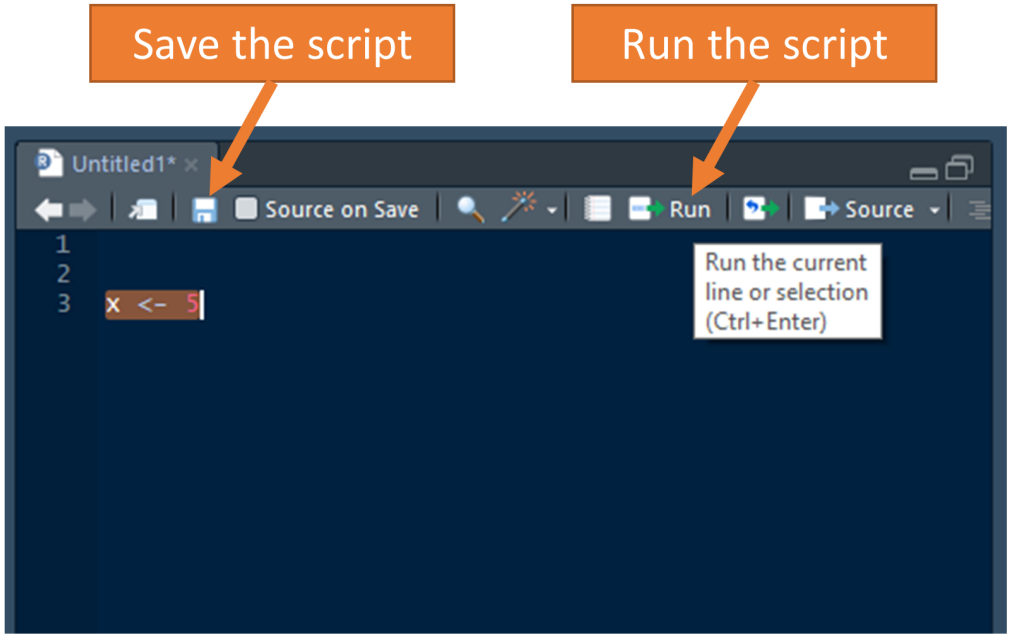
Creating a New Script

- In **R**, we can type our commands in the console; but once we close **R**, everything we have typed will be gone. Therefore, we should create an empty script, write the codes in the script, and save it for future use.
- The **R** script file has the .R extension, but it is essentially a text file. Thus, any text editor (e.g., Microsoft Word) can be used to open a script file.



Using the Script

When we type some codes in the script, we can select the lines we want to run and then hit the run button.

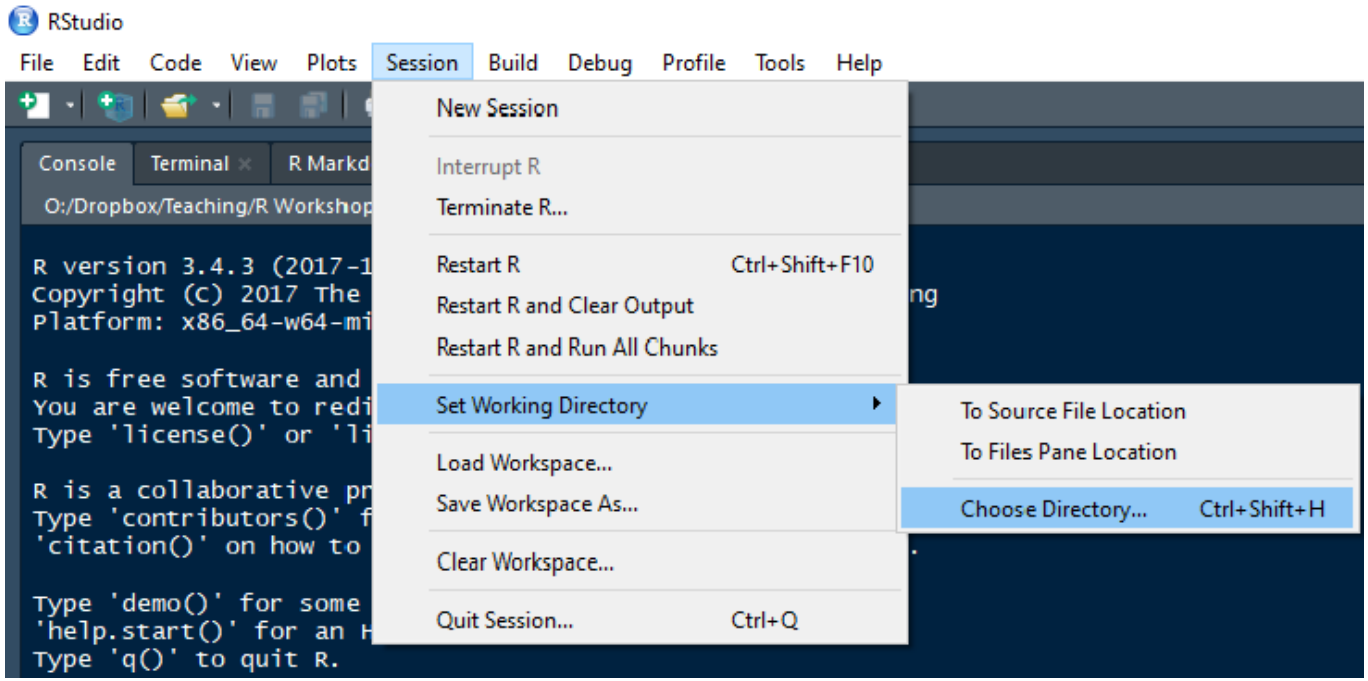


Alternatively, we can bring the cursor at the beginning of the line and hit the run button (which runs only a single line).

Working Directory

- An important feature of **R** is "working directory", which refers to a location or a folder in your computer where you keep your **R** script, your data files, etc.
- Once we define a working directory in **R**, any data file or script within that directory can be easily imported into **R** without specifying where the file is located.
- We can set the working directory in two ways:
  1. Using the "Session" options menu in **RStudio**
  2. Using the `setwd` command in the console

Working Directory



Working Directory

- Tpying the following code in the console will set the "R workshop" folder on my desktop as the working directory. If the folder path is correct, **R** changes the working directory without giving any error messages in the console.

```
setwd("C:/Users/bulut/Desktop/R workshop")
```

- To ensure that the working directory is properly set, we can use the `getwd` command:

```
getwd()
```

- Note that **R** does not accept any backslashes in the file path. Instead of a backslash, we need to use a frontslash. This is particulary important for Windows computers since the file paths involve backslashes (Mac OS X doesn't have this problem).

Installing Packages

- The base **R** program comes with many built-in functions to compute a variety of statistics and to create graphics (e.g., histograms, scatterplots, etc.)
- What makes **R** more powerful than other software programs is that **R** users can write functions and share them with other **R** users via the CRAN website.
- For example, **ggplot2** is a well-known **R** package, created by Hadley Wickham and Winston Chang. This package allows users to create elegant data visualizations. To download and install the **ggplot2** package, we need to use the `install.packages` command.

```
install.packages("ggplot2")
```

Installing Packages

- Once a package is downloaded and installed, it is permanently in your **R** folder (no need to re-install it).
- Although a package is already in the **R** folder, it is not accessible until we activate it.
- Whenever we need to access a package in **R**, the `library` command should be used.
- For example, to access the **ggplot2** package, we would use:

```
library("ggplot2")
```

Exercise (1)

- Create a new folder on your desktop and name it as "my folder".
- Open **RStudio** and set "my folder" on the desktop as your working directory using either the `setwd` command or the Session options menu in **RStudio**.
- Next, install and activate the **lattice** package using the `install.packages` and `library` commands. The **lattice** package is another important package for data visualization in **R**.

```
install.packages("lattice")
library("lattice")
```

Simple Calculations in R

R can be used like a calculator to do addition, subtraction, multiplication, and division. For example, if you type the following in the console, you should see the results in the same window.

```
6 + 20
[1] 26
89 - 53
[1] 36
424 * 68
[1] 28832
1110/37
[1] 30
20 + (30 * 3)
[1] 110
(20/5) + (144/12)
[1] 16
```

More Complex Calculations in R

Relatively more complex calculations are also possible in **R**.

```
10^2
[1] 100
sqrt(81)
[1] 9
log(5)
[1] 1.609438
exp(1.5)
[1] 4.481689
cos(100)
[1] 0.8623189
sin(180)
[1] -0.8011526
```

Creating Variables in R

To create a new variable in **R**, we use the assignment operator, "<-". To create a variable "x" that equals 1, we need to type:

```
x <- 1
```

If we want to print x, we just type "x" in the console and hit enter. **R** returns the value assigned to x.

```
x
```

```
[1] 1
```

We can also create a variable that holds multiple values in it, using the "c" command (c standards for "combine").

```
weight <- c(60, 72, 80, 84, 56)
weight
```

```
[1] 60 72 80 84 56
```

```
height <- c(1.7, 1.75, 1.8, 1.9, 1.6)
height
```

```
[1] 1.70 1.75 1.80 1.90 1.60
```

Creating Variables in R

Once we create a variable, we can do further calculations with it. Let's say we want to transform the weight variable (in kg) to a new variable called weight2 (in lbs).

```
weight2 <- weight * 2.20462
weight2
```

```
[1] 132.2772 158.7326 176.3696 185.1881 123.4587
```

Note that we named the variable as "weight2". So, both weight and weight2 exist in the active **R** session now. If we used the following, this would overwrite the existing "weight" variable.

```
weight <- weight * 2.20462
```

We can define a new variable based on the existing variables.

```
reading <- c(80, 75, 50, 44, 65)
math <- c(90, 65, 60, 38, 70)
total <- reading + math
total
```

```
[1] 170 140 110  82 135
```

Creating Variables in R

Sometimes we need a variable that holds character strings rather than numerical values. If a value is not numerical, we need to use double quotation marks. In the example below, we create a new variable called "cities" that has four city names in it. Each city name is written with double quotation marks.

```
cities <- c("Edmonton", "Calgary", "Red Deer", "Spruce Grove")
cities
```

```
[1] "Edmonton"     "Calgary"      "Red Deer"     "Spruce Grove"
```

We can also treat numerical values as character strings. For example, assume that we have a gender variable where 1=Male and 2=Female. We want **R** to know that these values are not actual numbers; instead, they are just numerical labels for gender groups.

```
gender <- c("1", "2", "2", "1", "2")
gender
```

```
[1] "1" "2" "2" "1" "2"
```

Some Rules...

R is case-sensitive!

R codes written in lowercase would **NOT** refer to the same codes written in uppercase.

```
cities <- c("Edmonton", "Calgary", "Red Deer", "Spruce Grove")
Cities
CITIES
Error: object 'Cities' not found
Error: object 'CITIES' not found
```

Variable names in **R**

- A variable name **CAN'T** begin with a number.
- A variable name **CAN'T** include a space.

```
4cities <- c("Edmonton", "Calgary", "Red Deer", "Spruce Grove")
my cities <- c("Edmonton", "Calgary", "Red Deer", "Spruce Grove")
Error: unexpected symbol in "4cities"
Error: unexpected symbol in "my cities"
```

Some Rules...

Naming conventions in **R**

- All lowercase: e.g. mycities
- Period.separated: e.g. my.cities
- Underscore_separated: e.g. my_cities
- Numbers at the end: e.g. mycities2018
- Combination of some of these rules: my.cities.2018

- Not to create messy code that is difficult to read and understand, I recommend using consistent and clear naming conventions.
- I personally prefer all lowercase with underscore (e.g., my_variable).

Some Rules...

Commenting in **R**

The hashtag symbol (#) is used for commenting in **R**. Any words, codes, etc. coming after a hashtag are just ignored.

```
# Here I define four cities in Alberta
cities <- c("Edmonton", "Calgary", "Red Deer", "Spruce Grove")
```

- I strongly recommend using comments throughout your codes.
- These annotations would remind you what you did and why you did it that way.
- You can easily comment out a line without having to remove it from your codes.

```
# cities <- c('Edmonton', 'Calgary', 'Red Deer', 'Spruce
# Grove')
cities <- c("Edmonton", "Calgary", "Red Deer")
```

Exercise (2)

- Open a new **R** script using File –> New File –> R Script.
- Create two new variables mpg (mile per gallon) and hp (horsepower) for five cars:
  - mpg: 21, 21, 22.8, 21.4, 18.7
  - hp: 110, 110, 93, 110, 175
- Finally, type the following code in your script and run it to find the correlation between mgp and hp:

```
cor(mpg, hp)
```

Creating Data Sets in R

Okan Bulut ©
bulut@ualberta.ca

There are multiple ways of creating data sets in **R**. We can create individual variables and combine them using the cbind (column bind) command:

```
mpg <- c(21, 21, 22.8, 21.4, 18.7)
hp <- c(110, 110, 93, 110, 175)
mydata <- cbind(mpg, hp)
mydata
```

```
      mpg  hp
[1,] 21.0 110
[2,] 21.0 110
[3,] 22.8  93
[4,] 21.4 110
[5,] 18.7 175
```

Creating Data Sets in R

Okan Bulut ©
bulut@ualberta.ca

We can create individual rows and combine them using the rbind (row bind) command (quite tedious if there are many rows):

```
car1 <- c(21, 110)
car2 <- c(21, 110)
car3 <- c(22.8, 93)
car4 <- c(21.4, 110)
car5 <- c(18.7, 175)
mydata <- rbind(car1, car2, car3, car4, car5)
mydata
```

```
     [,1] [,2]
car1 21.0  110
car2 21.0  110
car3 22.8   93
car4 21.4  110
car5 18.7  175
```

Creating Data Sets in R

Okan Bulut ©
bulut@ualberta.ca

We can define variables within a data set (called data frame in **R**):

```
mydata <- data.frame(mpg = c(21, 21, 22.8, 21.4, 18.7), hp = c(110,
    110, 93, 110, 175))
mydata
```

```
   mpg  hp
1 21.0 110
2 21.0 110
3 22.8  93
4 21.4 110
5 18.7 175
```

Data frames in **R** are very convenient because many mathematical operations can be directly applied to a data frame or some columns (or rows) of a data frame. Once a data set is defined, we can see its content using the View command (which open the data window) or the head command (which prints the first six rows of the data):

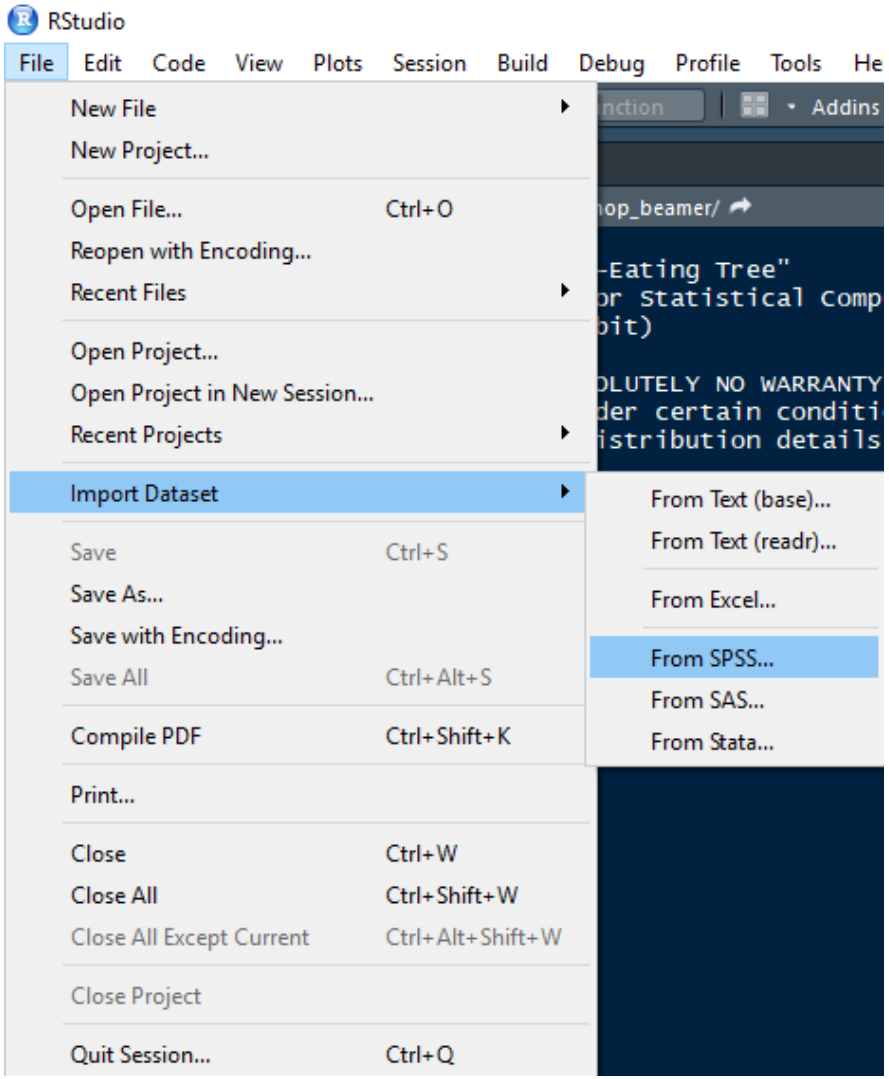
```
View(mydata)
head(mydata)
```

Importing Data into R

- We often save our data sets in convenient data formats, such as Excel, SPSS, or text files (.txt, .csv, .dat, etc.).
- **R** is capable of importing (i.e., reading) various data formats.
- There are two possible ways to read a data file in **R**:
  1. By using the "Import Dataset" menu option
  2. By reading the file with an **R** command
     - **R** has some built-in functions, such as read.csv and read.table
     - There are some **R** packages for specific data formats; "foreign" for SPSS files and "xlsx" for Excel files

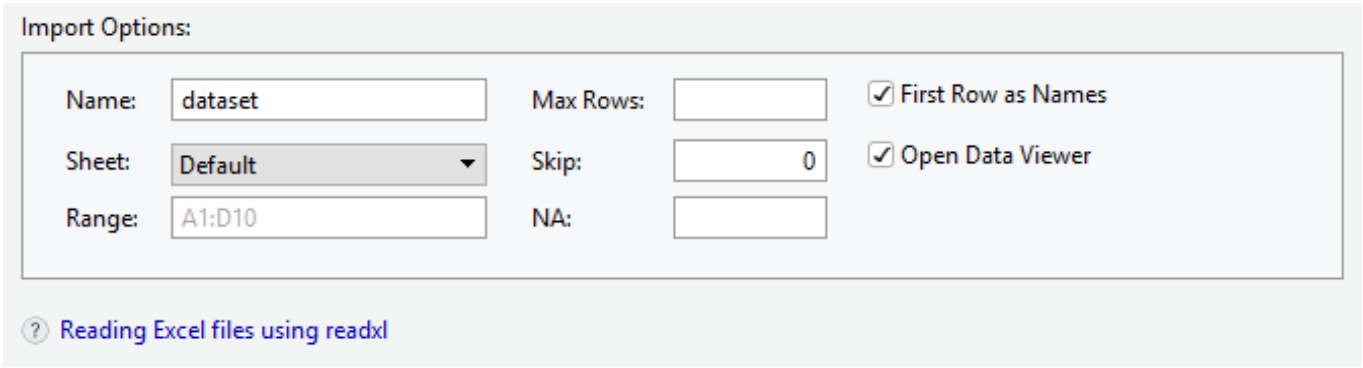
Importing Data into R

Okan Bulut ©
bulut@ualberta.ca



Importing Excel Files

- Browse the file that you want to import
- Give a name for the data set
- Choose the sheet to be imported
- "First Row as Names" if the variable names are in the first row



Importing SPSS Files

- Browse the file that you want to import
- Give a name for the data set
- Choose the SPSS data format (SAV)

Importing Data Files with R Commands

Excel files:

```
install.packages("xlsx")
library("xlsx")
my_excel_file <- read.xlsx("path to the file/filename.xlsx",
    sheetName = "sheetname")
```

SPSS files:

```
install.packages("foreign")
library("foreign")
my_spss_file <- read.spss("path to the file/filename.sav", to.data.frame = TRUE)
```

Text files:

```
my_csv_file <- read.csv("path to the file/filename.csv", header = TRUE)
my_txt_file <- read.table("path to the file/filename.txt", header = TRUE,
    sep = "\t")
```

- `header=TRUE` if the variable names are in the first row
- `sep="\t"` for tab-separated files; `sep=","` for comma-separated files

Exercise (3)

- In the workshop materials, you will find three data files:
  - `interest.xlsx` (Excel)
  - `interest.sav` (SPSS)
  - `interest.csv` (Comma Separated Values)
- The `interest` dataset comes from a fabricated cognitive, personality, and vocational interest inventory.
- Open **RStudio** and set your working directory to where you downloaded the workshop materials.
- Import the files by using either the "Import Dataset" menu option or the **R** commands.

```
library("foreign")
library("xlsx")
interest_csv <- read.csv("interest.csv", header = TRUE)
interest_xlsx <- read.xlsx("interest.xlsx", sheetName = "interest")
interest_spss <- read.spss("interest.sav", to.data.frame = TRUE)
```

Viewing and Indexing Data

- Once a dataset is imported into **R**, we should ensure that **R** was able to read the data properly. The `head` command prints out the first six rows of the dataset.

```
interest <- read.csv("interest.csv", header = TRUE)
head(interest)
```

- To see how many columns and rows exist in the dataset, we can use the `dim` (i.e., dimension) command:

```
dim(interest)
```

```
[1] 250  20
```

- The interest dataset has 250 rows (i.e., individuals) and 20 rows (i.e., variables).

Viewing and Indexing Data

- The `str` command prints the content of the dataset.

```
str(interest)
```

```
'data.frame': 250 obs. of  20 variables:
 $ id       : int  1 2 3 4 5 6 7 8 9 10 ...
 $ gender   : int  2 1 2 2 1 1 1 1 1 2 ...
 $ gender2  : Factor w/ 2 levels "Female","Male": 2 1 2 2 1 1 1 1 1 2 ...
 $ educ     : int  18 12 12 12 14 14 12 12 12 12 ...
 $ age      : int  47 39 50 41 33 26 46 42 28 59 ...
 $ vocab    : num  66.7 38.1 65.6 44.1 51.5 62.1 50.9 59.5 40.7 54.7 ...
 $ reading  : num  66.7 35.7 61.1 49.7 59.8 60.4 49.6 54.9 37.5 53.8 ...
 $ mathmtcs: num  59 48.3 64 50.5 52 49.9 46.9 58.1 40.2 49.7 ...
 $ analyrea: num  66.5 52.3 70.4 48.3 55.8 46.3 56.3 54.3 38.4 51.4 ...
 $ sociabty: num  61.1 38.5 59 58.7 42.1 66.4 48.2 60 49.8 44.2 ...
 $ stress   : num  52.3 53.7 57.3 45.7 40.6 30.1 55.7 43.6 43.6 56.1 ...
 $ impulsve: num  42.2 52.9 41.8 49.4 51.2 51.3 58.4 48.1 61.2 58.2 ...
 $ policemn: num  25.8 47.3 53.3 55.8 77 45.7 52.2 48.2 62.4 44.2 ...
 $ salesrep: num  50.8 53.8 53.2 49.7 36.7 38.8 44.6 40.7 58 56.1 ...
 $ teacher : num  74.8 48 52.7 43.5 41.7 57.7 41.2 63.8 53 69.1 ...
 $ artist   : num  68 39.6 50.9 53.5 46.5 53.7 44.5 44.8 44.5 62.5 ...
 $ socworkr: num  82.3 37.4 63.1 54.9 34.2 59 40.5 53 51.6 64.4 ...
 $ doctor   : num  69.5 45 56.9 49.3 53.5 60.4 47 50 51.6 69.4 ...
 $ actor    : num  61.3 22.7 55.1 38.4 37.2 81.6 43.7 48.7 41.3 55.7 ...
 $ lawyer   : num  61.6 45.8 61.2 57.8 54.5 65.6 48.1 58.6 58.7 51 ...
```

Viewing and Indexing Data

- If we want to see a specific variable, we can either specify the dataset name (followed by $) and the variable name.
- For example, we can print the first six rows of the variable "gender2" as follows:

```
head(interest$gender2)
```

```
[1] Male   Female Male   Male   Female Female
Levels: Female Male
```

- It is also possible to view multiple variables together.

```
head(interest[, c("gender2", "age")])
```

```
  gender2 age
1    Male  47
2  Female  39
3    Male  50
4    Male  41
5  Female  33
6  Female  26
```

Viewing and Indexing Data

- In **R**, square brackets ([...]) are used for indexing. Within the square brackets, the first number shows the row number(s) and the second number shows the column(s).
- For example; if we wanted to see the second variable for the fifth person in the dataset, then we would do:

```
interest[5, 2]
```

```
[1] 1
```

- Or, if we wanted to see the first three variables for the first five persons, we would do:

```
interest[1:5, 1:3]
```

```
  id gender gender2
1  1      2    Male
2  2      1  Female
3  3      2    Male
4  4      2    Male
5  5      1  Female
```

Viewing and Indexing Data

- Instead of `interest[1:5,1:3]`, we could also use:

```
interest[c(1, 2, 3, 4, 5), c(1, 2, 3)]
```

```
  id gender gender2
1  1      2    Male
2  2      1  Female
3  3      2    Male
4  4      2    Male
5  5      1  Female
```

or

```
interest[1:5, c("id", "gender", "gender2")]
```

```
  id gender gender2
1  1      2    Male
2  2      1  Female
3  3      2    Male
4  4      2    Male
5  5      1  Female
```

Subsetting Data

- Assume that we want to create a new dataset with only females from the interest dataset. We can subset the females in two ways:

```
interest_female <- subset(interest, gender2 == "Female")
```

or

```
interest_female <- interest[interest$gender2 == "Female", ]
```

- Both options make a conditional selection where gender2 is equal to "Female" ("==" makes a conditional selection).
- Another example of subsetting with gender2 and age:

```
female_50 <- subset(interest, gender2 == "Female" & age < 50)
```

Recoding Variables

Okan Bulut ©
bulut@ualberta.ca

- There are multiple ways of recoding variables in **R**.
- For a binary recoding, the `ifelse` command is very easy to use.
- Assume that we want to create a new education variable identifying those with a college degree based on the years of education:

```
interest$educ2 <- ifelse(interest$educ > 12, "College", "High School")
interest$gender_binary <- ifelse(interest$gender == "Female", 1, 0)
```

- For more complex recoding, the **car** package offers a useful recode function:

```
install.packages("car")
library("car")
interest$educ2 <- recode(interest$educ, "13:18='College'; 8:12='High School'")
interest$gender_binary <- recode(interest$gender2, "'Female'=1; 'Male'=0")
```

Summarizing Data

- In **R**, there are many ways for summarizing the data.
- The `summary` command returns the min and max, mean, median, and first and third quartiles for numerical variables and frequencies for categorical variables.
- Using the same interest dataset, we can summarize five variables (gender2, age, educ, vocab, reading):

```
summary(interest[, c("gender2", "age", "educ", "vocab", "reading")])
```

```
   gender2          age              educ           vocab           reading
 Female:128   Min.   :11.00   Min.   : 8.0   Min.   :23.80   Min.   :25.30
 Male  :122   1st Qu.:33.00   1st Qu.:12.0   1st Qu.:43.95   1st Qu.:44.83
              Median :40.00   Median :12.0   Median :50.40   Median :51.85
              Mean   :39.55   Mean   :12.3   Mean   :50.90   Mean   :51.35
              3rd Qu.:46.00   3rd Qu.:12.0   3rd Qu.:58.60   3rd Qu.:57.98
              Max.   :65.00   Max.   :18.0   Max.   :76.30   Max.   :77.00
```

Frequency Tables

- For categorical variables, the `table` command can be used for creating frequency tables:

```
table(interest$gender2)

Female   Male
   128    122
table(interest$educ2)

    College High School
         34         216
table(interest$gender2, interest$educ2)

         College High School
  Female      15         113
  Male        19         103
```

Descriptive Statistics

Okan Bulut ©
bulut@ualberta.ca

- We can calculate mean, median, variance, standard deviation, minimum, and maximum for individual variables.

```
mean(interest$age)
[1] 39.548
median(interest$age)
[1] 40
var(interest$age)
[1] 100.835
sd(interest$age)
[1] 10.04167
min(interest$age)
[1] 11
max(interest$age)
[1] 65
```

Descriptive Statistics by Group

- We can also calculate the descriptive statistics for each value of a categorical variable (e.g., gender) using the `tapply` command.

```
# Mean age by gender
tapply(interest$age, interest$gender2, mean)
```

```
  Female     Male
39.21094 39.90164
```

```
# Median age by education
tapply(interest$age, interest$educ2, median)
```

```
    College High School
         37          41
```

```
# Standard deviation of reading score by gender
tapply(interest$reading, interest$gender2, sd)
```

```
   Female      Male
 9.746818 10.038121
```

```
# Variance of reading score by education
tapply(interest$reading, interest$educ2, var)
```

```
    College High School
   46.46592    88.42043
```

Descriptive Statistics by Group

- For a detailed summary of variables, I strongly recommend the **skimr** package.

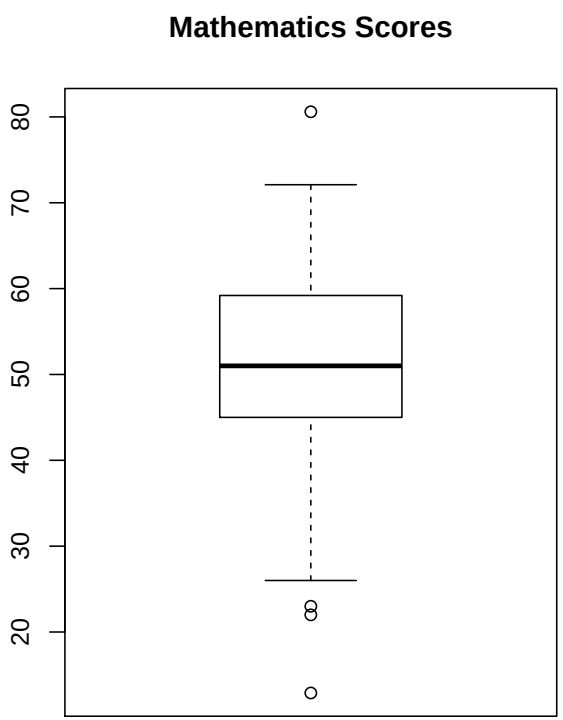
```
install.packages("skimr")
library("skimr")
skim(interest[, c("age", "educ", "reading", "vocab", "mathmtcs")])
```

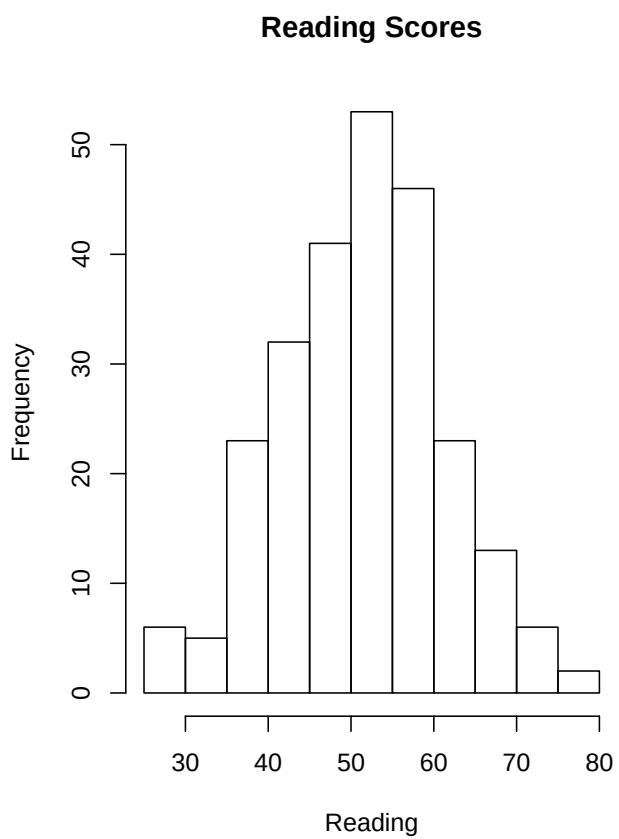
```
Skim summary statistics
 n obs: 250
 n variables: 5

Variable type: integer
 variable missing complete   n  mean    sd p0 p25 median p75 p100
      age       0      250 250 39.55 10.04 11  33     40  46   65
     educ       0      250 250 12.3   1.61  8  12     12  12   18

Variable type: numeric
 variable missing complete   n  mean    sd   p0   p25 median   p75 p100
 mathmtcs       0      250 250 51.05 10.54 12.9 45.08  51    59.2  80.6
  reading       0      250 250 51.35  9.91 25.3 44.82  51.85 57.98 77
    vocab       0      250 250 50.9   9.98 23.8 43.95  50.4  58.6  76.3
```

Okan Bulut ©
bulut@ualberta.ca

- Boxplots and histograms

```
boxplot(interest$mathmtcs, main = "Mathematics Scores")
hist(interest$reading, main = "Reading Scores", xlab = "Reading")
```





Okan Bulut ©
bulut@ualberta.ca

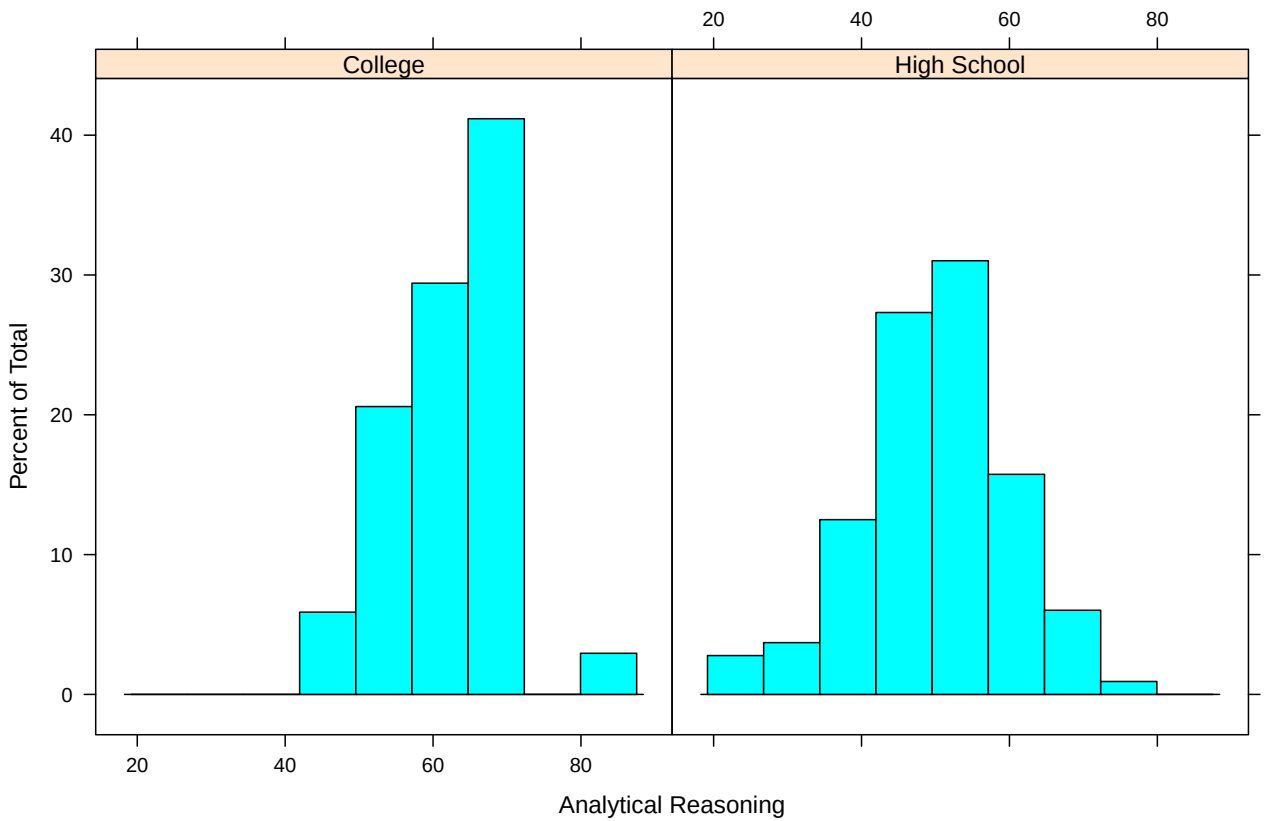
- Boxplots by group

```
boxplot(interest$stress ~ interest$gender2, xlab = "Gender", ylab = "Stress",
    main = "Stress Level by Gender", names = c("Female", "Male"))
```

Base Graphics in R

- Histograms by group (with lattice)

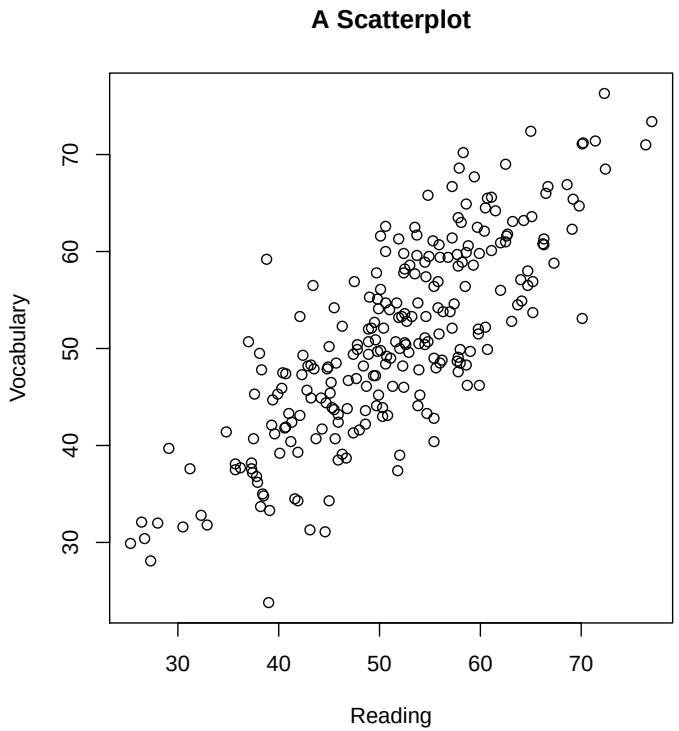
```
library("lattice")
histogram(~analyrea | educ2, data = interest, xlab = "Analytical Reasoning")
```

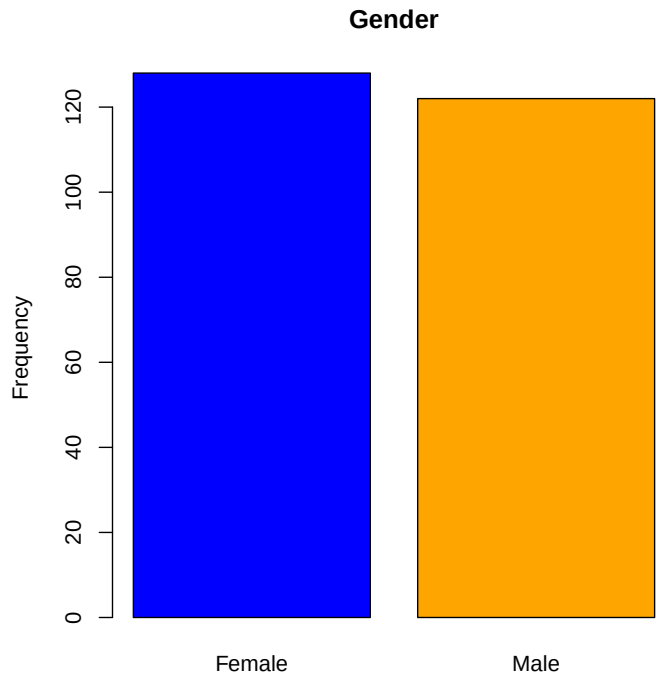


Okan Bulut ©
bulut@ualberta.ca

- Scatterplots and bar plots

```
plot(interest$reading, interest$vocab, main = "A Scatterplot", xlab = "Reading",
    ylab = "Vocabulary")
barplot(table(interest$gender2), main = "Gender", names = c("Female", "Male"),
    ylab = "Frequency", col = c("blue", "orange"))
```

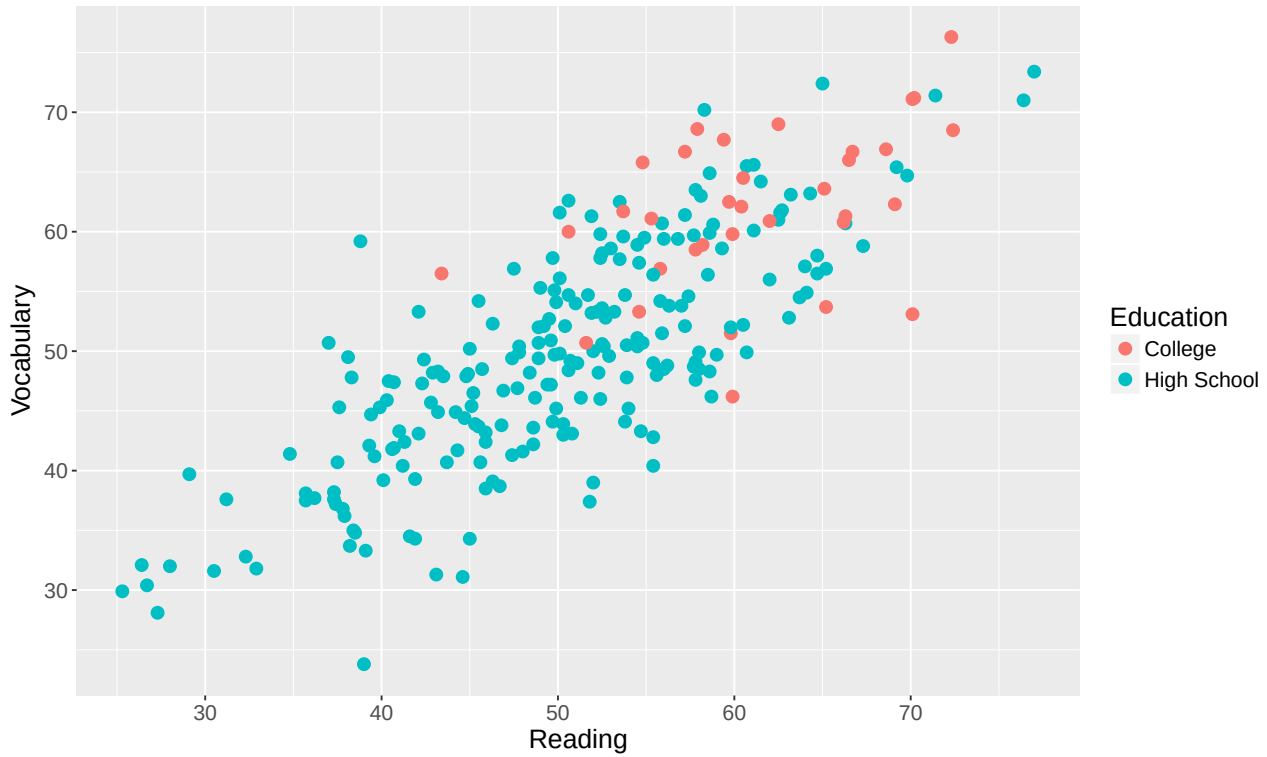




Graphics with ggplot2

- The **ggplot2** package is capable of creating more elegant and complex graphics (check out http://ggplot2.tidyverse.org/).

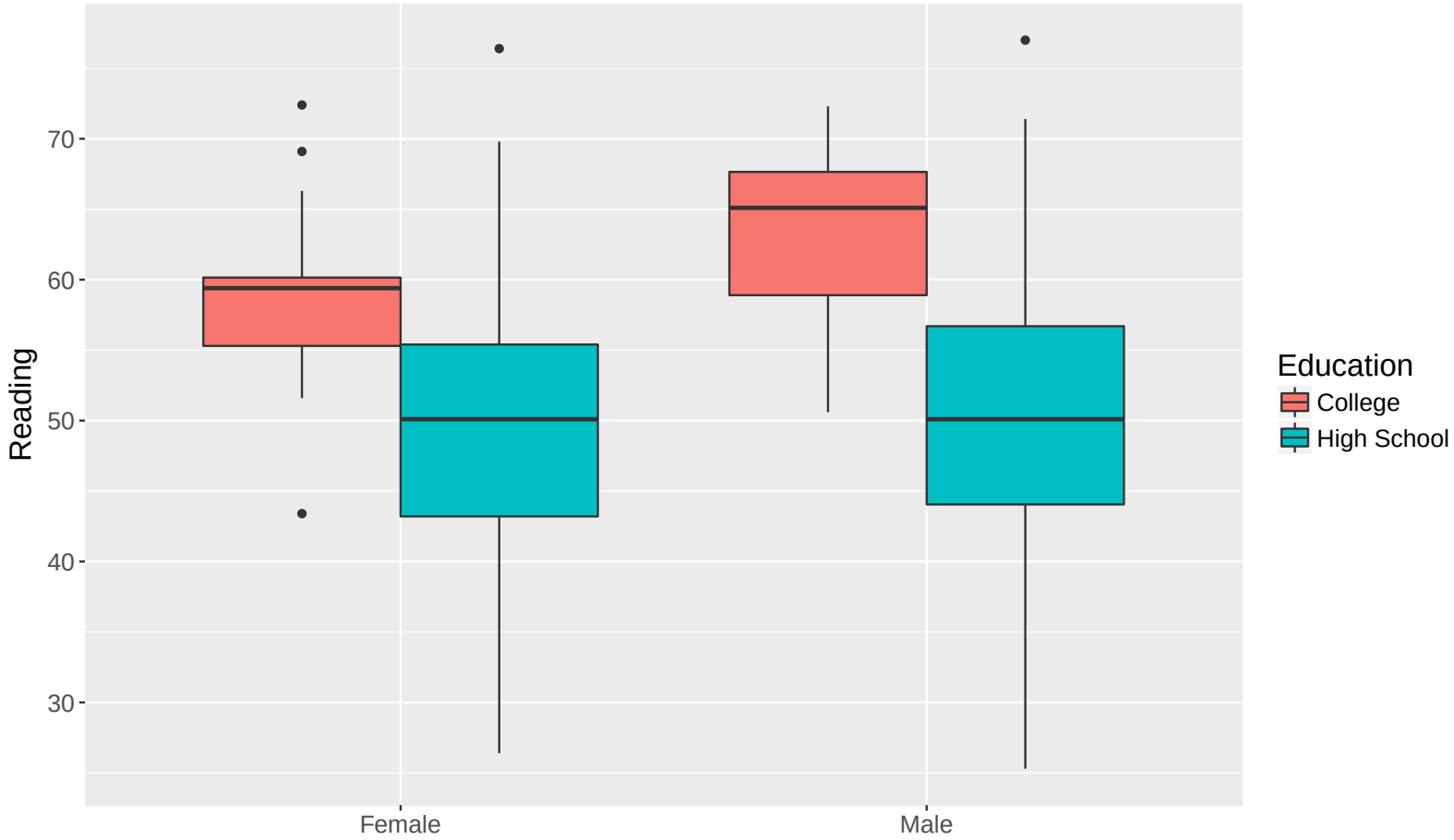
```
library("ggplot2")
ggplot(interest, aes(reading, vocab, colour = educ2)) + geom_point(size = 3) +
    labs(colour = "Education", x = "Reading", y = "Vocabulary")
```



Okan Bulut ©
bulut@ualberta.ca

Graphics with ggplot2

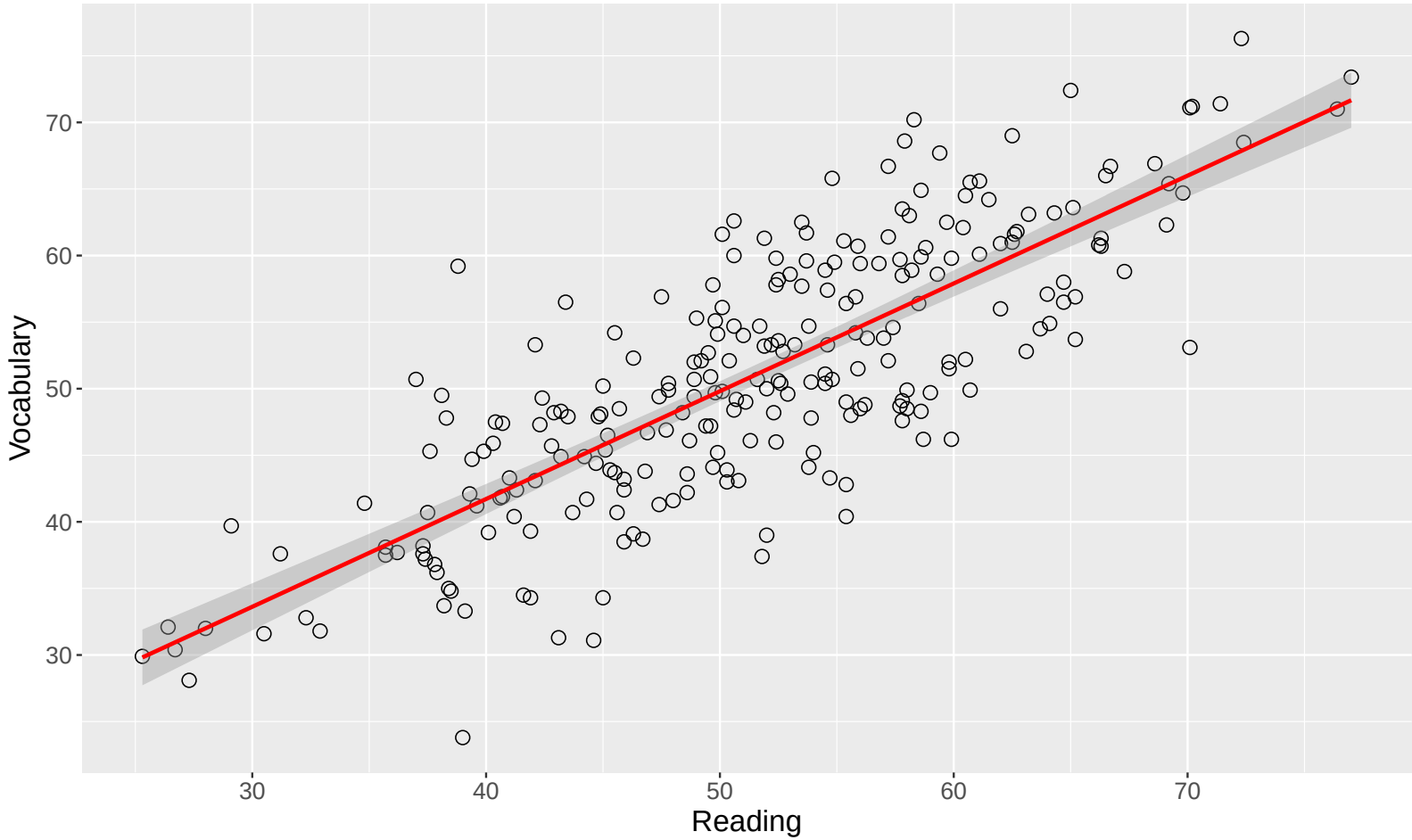
```
ggplot(interest, aes(gender2, reading, fill = educ2)) + labs(x = "",
    y = "Reading", fill = "Education") + geom_boxplot()
```



Okan Bulut ©
bulut@ualberta.ca

Graphics with ggplot2

```
ggplot(interest, aes(reading, vocab)) + geom_point(shape = 1, size = 3) +
    geom_smooth(method = lm, color = "red", se = TRUE) + labs(x = "Reading",
    y = "Vocabulary")
```



Inferential Statistics

With hypothesis testing, we can calculate several inferential statistics:

1. Whether a sample mean is equal to a particular value:
   - One sample t test ($H_0 : \mu = \text{value}$)
2. Whether two sample means are equal:
   - Independent samples t test ($H_0 : \mu_1 = \mu_2$ or $H_0 : \mu_1 - \mu_2 = 0$)
   - Repeated measures t test ($H_0 : \mu_D = 0$ or $H_0 : \mu_1 - \mu_2 = 0$)
3. Whether three or more groups have equal means:
   - ANOVA ($H_0 : \mu_1 = \mu_2 = \mu_3 = \cdots = \mu_k$)
4. Whether two variables are correlated

One Sample t Test

Hypothesis: Does the average reading score in our sample differ from the average reading score ($\mu = 50$) in the population?

```
t.test(interest$reading, mu = 50, conf.level = 0.95, alternative = "two.sided")
```

```
One Sample t-test

data:  interest$reading
t = 2.1543, df = 249, p-value = 0.03218
alternative hypothesis: true mean is not equal to 50
95 percent confidence interval:
 50.11575 52.58345
sample estimates:
mean of x
  51.3496
```

Conclusion: With the significance level of $\alpha = .05$, we reject the null hypothesis that the average reading score in the interest dataset is the same as the average reading score in the population, $t(249) = 2.15$, $p < .05$, $CI_{95} = [50.12, 52.58]$.

Independent Samples t Test

Hypothesis: Does the average analytical reasoning score differ between males and females?

```
male <- interest[interest$gender2 == "Male", "analyrea"]
female <- interest[interest$gender2 == "Female", "analyrea"]
t.test(male, female, conf.level = 0.95, alternative = "two.sided")
```

```
Welch Two Sample t-test

data:  male and female
t = -1.5327, df = 246.18, p-value = 0.1266
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -4.6752628  0.5833058
sample estimates:
mean of x mean of y
 50.70246  52.74844
```

Conclusion: With the significance level of $\alpha = .05$, we fail to reject the null hypothesis that the average analytical reasoning score differ between males and females, $t(246) = -1.53$, $p = 0.1266$.

Analysis of Variance (ANOVA)

Hypothesis: Do different age groups differ in analytical reasoning?

```
# Creating age2 as a categorical age variable
interest$age2[interest$age < 35] <- "Group 1"
interest$age2[interest$age >= 35 & interest$age < 45] <- "Group 2"
interest$age2[interest$age >= 45] <- "Group 3"
table(interest$age2)


Group 1 Group 2 Group 3
     77      93      80
anova(lm(interest$analyrea ~ interest$age2))
Analysis of Variance Table

Response: interest$analyrea
               Df  Sum Sq Mean Sq F value Pr(>F)
interest$age2   2   449.2  224.61  2.0272 0.1339
Residuals     247 27366.4  110.80
```

Conclusion: With the significance level of $\alpha = .05$, we fail to reject the null hypothesis that analytical reasoning differs between the three age groups.

Correlations

Are reading scores and vocabulary scores correlated with each other?

```
cor(interest$vocab, interest$reading, method = "pearson")
```

```
[1] 0.8030912
```

To test the significance of the correlation:

```
install.packages("Hmisc")
library("Hmisc")
rcorr(interest$vocab, interest$reading, type = "pearson")
```

```
    x   y
x 1.0 0.8
y 0.8 1.0

n= 250


P
  x  y
x     0
y  0
```

Conclusion: The correlation between the reading and vocabulary scores is statistically significant, $r = .80$, p-value $< .001$.

THANK YOU!

For questions/comments: bulut@ualberta.ca