Week 1: Getting Started with R

EMSE 4574: Intro to Programming for Analytics

John Paul Helveston

September 01, 2020
Week 1: *Getting Started with R*

1. Course Introduction
2. Break: Install Course Tools
3. Getting started with R & RStudio
4. Operators & data types
5. Preview of HW 1
Week 1: *Getting Started with R*

1. **Course Introduction**
2. Break: Install Course Tools
3. Getting started with R & RStudio
4. Operators & data types
5. Preview of HW 1
Meet your instructor!

John Helveston, Ph.D.

Assistant Professor, Engineering Management & Systems Engineering

- 2016-2018 Postdoc at Institute for Sustainable Energy, Boston University
- 2016 PhD in Engineering & Public Policy at Carnegie Mellon University
- 2015 MS in Engineering & Public Policy at Carnegie Mellon University
- 2010 BS in Engineering Science & Mechanics at Virginia Tech
- Website: www.jhelvy.com
Meet your tutors!

Saurav Pantha (aka "The Firefighter")

- Graduate Assistant (GA)
- Masters student in EMSE
Meet your tutors!

**Jennifer Kim** (aka "The Monitor")

- Learning Assistant (LA)
- EMSE Junior & P4A alumni
Course orientation

🌍 Everything you need will be on the course website:
https://p4a.seas.gwu.edu/2020-Fall/

🛠 Course is broken into **two chunks**:

1. Programming

2. Analytics
Homeworks (48% of grade)

每一天 (12 total)

Soft due dates (11pm Monday before class)

⚠️ Don't abuse this flexibility

Two hard deadlines on homework submissions:

1. Oct. 20 (HWs 1-6)
2. Dec. 08 (HWs 7-12)
Quizzes (15% of grade)

- In class every other week-ish (7 total, drop lowest 2)
- 5 minutes (3-5 questions)

Example quiz

Why quiz at all? There's a phenomenon called the "retrieval effect" - basically, you have to *practice* remembering things, otherwise your brain won't remember them (details in the book *Make It Stick: The Science of Successful Learning*).
Exams (32% of grade)

Midterm (weeks 1 - 6) on Oct. 20

Final (weeks 1 - 13) on Dec. 15
# Grading: Standard

<table>
<thead>
<tr>
<th>Course Component</th>
<th>Weight</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homeworks</td>
<td>48%</td>
<td>12 x 4% each</td>
</tr>
<tr>
<td>Quizzes</td>
<td>15%</td>
<td>5 x 3% each</td>
</tr>
<tr>
<td>Midterm Exam</td>
<td>12%</td>
<td></td>
</tr>
<tr>
<td>Final Exam</td>
<td>20%</td>
<td></td>
</tr>
<tr>
<td>Participation</td>
<td>5%</td>
<td></td>
</tr>
</tbody>
</table>
Grading: Alternative Minimum Grade (AMG)

- Students who struggle early on, but work hard to succeed in 2nd half.
- Highest possible grade is "C"

<table>
<thead>
<tr>
<th>Course Component</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best 10 Homeworks</td>
<td>40%</td>
</tr>
<tr>
<td>Best 4 Quizzes</td>
<td>10%</td>
</tr>
<tr>
<td>Midterm Exam</td>
<td>10%</td>
</tr>
<tr>
<td>Final Exam</td>
<td>40%</td>
</tr>
</tbody>
</table>
Course policies

- BE NICE. BE HONEST. DON'T CHEAT.
- Write your own code (even in "collaborative" assignments)
- Don't cheat
How to succeed in this class

Take care of your brain

- Sleep!
- Exercise!
- Eat good food!

Start HW early!

Take breaks often!

Ask for help!
Getting Help

⭐ Use Slack to ask questions.

👩‍💻 Meet with your tutors

학생 메시지 Schedule a meeting w/ Prof. Helveston:

- Tuesdays from 3:30-4:30pm
- Wednesdays from 2:00-4:30pm
- Fridays from 12:00-2:00pm

📞 GW Coders
Course Tools (see course prep lesson)

Slack

- Link to join (also posted on Blackboard announcement).
- Install Slack on your phone and turn notifications on!
Course Tools (see course prep lesson)

R & RStudio (Install both)

After installed:

Open this:

Not this:
Course Tools (see course prep lesson)

WiFi GWU VPN (Install Cisco AnyConnect VPN Client)

WiFi + R = RStudio online!
Week 1: *Getting Started with R*

1. Course Introduction
2. **Break: Install Course Tools**
3. Getting started with R & RStudio
4. Operators & data types
5. Preview of HW 1
Install Course Tools (see course prep lesson)

- Slack
  - Link to join (also posted on Blackboard announcement).
  - Install Slack on your phone and turn notifications on!

- R & RStudio (Install both)

- GWU VPN (Install Cisco AnyConnect VPN Client)

+ = RStudio online!
Week 1: *Getting Started with R*

1. Course Introduction
2. Break: Install Course Tools
3. Getting started with R & RStudio
4. Operators & data types
5. Preview of HW 1
RStudio Orientation

- Know the boxes
- Customize the layout
- Customize the look
- Extra themes

Source

Environment/History

Console/Terminal

Files/Plots/Pkgs/Help
Your first conversation

Write stuff in the console, then press "enter"

Example: **addition**

```
3 + 4
```

## [1] 7

Example: **error**

```
3 + "4"
```

## Error in 3 + "4": non-numeric argument to binary operator
Storing values

Use the "<-" symbol to assign values to objects

Example:

```r
x <- 40
x
## [1] 40

x + 2
## [1] 42
```
Storing values

If you overwrite an object, R "forgets" the old value

Example:

```r
x <- 42
x
## [1] 42
x <- 50
x
## [1] 50
```

Storing values

You can also use the "=" symbol to assign values

(but you really should use "<-")

Example:

```
x = 42
x
```

```
## [1] 42
```

```
y <- 42
y
```

```
## [1] 42
```
Storing values

You can store more than just numbers

Example:

```r
x <- "If you want to view paradise"
y <- "simply look around and view it"

x
## [1] "If you want to view paradise"

y
## [1] "simply look around and view it"
```
### Storing values

#### Pro tip 1:

Shortcut for "<-" symbol

<table>
<thead>
<tr>
<th>OS</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>mac</td>
<td>option + -</td>
</tr>
<tr>
<td>windows</td>
<td>alt + -</td>
</tr>
</tbody>
</table>

(see [here](#) for more shortcuts)

#### Pro tip 2:

Always surround "<-" with spaces

Example:

```
x<-2
```

Does this mean `x <- 2` or `x < -2`?
R ignores **extra space**

```
x <- 2
y <- 3
z <- 4
```

Check:

```
x
## [1] 2

y
## [1] 3

z
## [1] 4
```

R cares about **case**

```
number <- 2
Number <- 3
numbeR <- 4
```

Check:

```
number
## [1] 2

Number
## [1] 3

numbeR
## [1] 4
```
Use # for comments

R ignores everything after the # symbol

Example:

```r
speed <- 42  # This is mph, not km/h!
speed
## [1] 42
```
Use meaningful variable names

**Example:** You are recording the speed of a car in mph

**Poor** variable name:

```r
x <- 42
```

**Good** variable name:

```r
speed <- 42
```

**Even better** variable name:

```r
car_speed_mph <- 42
```
Use standard casing styles

I recommend using one of these:

- `snake_case_uses_underscores`
- `camelCaseUsesCaps`

Example:

```
days_in_week <- 7
monthsInYear <- 12
```
The workspace

View all the current objects:

```
objects()
```

```
## [1] "car_speed_mph"  "days_in_week"  "monthsInY"
## [5] "Number"        "speed"         "x"
## [9] "y"             "z"
```

Remove an object by name:

```
rm(number)
```

```
objects()
```

```
## [1] "car_speed_mph"  "days_in_week"  "monthsInY"
## [5] "Number"        "speed"         "x"
## [9] "z"
```
View prior code in history pane

Use "up" arrow see previous code
Staying organized

1) Save your code in .R files
   File > New File > R Script

2) Keep work in R Project files
   File > New Project...
Your turn

A. Practice getting organized

1. Open RStudio and create a new R project called week1.
2. Create a new R script and save it as practice.R.
3. Open the practice.R file and write your answers to the question below in it.

B. Creating & working with objects

1). Create objects to store the values in this table:

<table>
<thead>
<tr>
<th>City</th>
<th>Area (sq. mi.)</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>San Francisco, CA</td>
<td>46.87</td>
<td>884,363</td>
</tr>
<tr>
<td>Chicago, IL</td>
<td>227.63</td>
<td>2,716,450</td>
</tr>
<tr>
<td>Washington, DC</td>
<td>61.05</td>
<td>693,972</td>
</tr>
</tbody>
</table>

2) Use the objects you created to answer the following questions:

- Which city has the highest density?
- How many _more_ people would need to live in DC?
Week 1: *Getting Started with R*

1. Course Introduction
2. Break: Install Course Tools
3. Getting started with R & RStudio
4. *Operators & data types*
5. Preview of HW 1
R as a calculator

Basic operators:

- Addition: +
- Subtraction: −
- Multiplication: *
- Division: /

Other important operators:

- Power: ^
- Integer Division: %/%
- Modulus: %%
Integer division: `%%`

Integer division drops the remainder

Example:

```
4 / 3 # Regular division
## [1] 1.333333

4 %% 3 # Integer division
## [1] 1
```
Integer division: `%/%`

Integer division drops the remainder

What will this return?

\[4 \%/% 4\]

## [1] 1

What will this return?

\[4 \%/% 5\]

## [1] 0
Modulus operator: `%%`

Modulus returns the remainder *after* doing integer division.

Example:

```
5 %% 3
## [1] 2
3.1415 %% 3
## [1] 0.1415
```
Modulus operator: %%

Modulus returns the remainder after doing integer division.

What will this return?

4 %% 4
## [1] 0

What will this return?

4 %% 5
## [1] 4
Odds and evens with $n \mod 2$

If $n \mod 2$ is $0$, $n$ is \textbf{EVEN}

10 \mod 2

12 \mod 2

Also works with negative numbers!

-42 \mod 2

If $n \mod 2$ is $1$, $n$ is \textbf{ODD}

1 \mod 2

13 \mod 2

Also works with negative numbers!

-47 \mod 2
Number "chopping" with 10s

The mod operator (%%) "chops" a number and returns everything to the right

```
123456 %% 1
## [1] 0

123456 %% 10
## [1] 6

123456 %% 100
## [1] 56
```

Integer division (%/%) "chops" a number and returns everything to the left

```
123456 %/% 1
## [1] 123456

123456 %/% 10
## [1] 12345

123456 %/% 100
## [1] 1234
```
Number "chopping" with 10s

- `%%` returns everything to the right ("chop" ->)
- `%/%` returns everything to the left (<- "chop")
- The "chop" point is always just to the right of the chopping digit:

<table>
<thead>
<tr>
<th>Example</th>
<th>&quot;Chop&quot; point</th>
<th>&quot;Chop&quot; point description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1234</td>
<td>Right of the 1’s digit</td>
</tr>
<tr>
<td>1234</td>
<td>123</td>
<td>Right of the 10’s digit</td>
</tr>
<tr>
<td>1234</td>
<td>12</td>
<td>Right of the 100’s digit</td>
</tr>
<tr>
<td>1234</td>
<td>1</td>
<td>Right of the 1,000’s digit</td>
</tr>
<tr>
<td>1234</td>
<td></td>
<td>Right of the 10,000’s digit</td>
</tr>
</tbody>
</table>
Comparing things: **Relational operators**

Compare if condition is **TRUE** or **FALSE** using:

- Less than: `<`
- Less than or equal to: `<=`
- Greater than or equal to: `>=`
- Greater than: `>`
- Equal: `==`
- Not equal: `!=`

<table>
<thead>
<tr>
<th>Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>2 &lt; 2</code></td>
<td># [1] FALSE</td>
</tr>
<tr>
<td><code>2 &lt;= 2</code></td>
<td># [1] TRUE</td>
</tr>
<tr>
<td><code>(2 + 2) == 4</code></td>
<td># [1] TRUE</td>
</tr>
<tr>
<td><code>(2 + 2) != 4</code></td>
<td># [1] FALSE</td>
</tr>
<tr>
<td>&quot;penguin&quot; == &quot;penguin&quot;</td>
<td># [1] TRUE</td>
</tr>
</tbody>
</table>
Comparing things: **Logical operators**

Make multiple comparisons with:

- And: 
- Or: |
- Not: !

With "and" (\&), every part must be **TRUE**, otherwise the whole statement is **FALSE**:

\[(2 \equiv 2) \& (3 \equiv 3)\]

## [1] TRUE

\[(2 \equiv 2) \& (2 \equiv 3)\]

## [1] FALSE
Comparing things: **Logical operators**

Make multiple comparisons with:

- And: `&`
- Or: `|`
- Not: `!`

With "or" (`|`), if *any* part is **TRUE**, the whole statement is **TRUE**:

\[
(2 == 2) | (3 == 3)
\]

## [1] **TRUE**

\[
(2 == 2) | (2 == 3)
\]

## [1] **TRUE**
Comparing things: **Logical operators**

Make multiple comparisons with:

- And: &
- Or: | (\[\text{OR}\] )
- Not: ! (\[\text{NOT}\] )

The "not" (\(!\) ) symbol produces the *opposite* statement:

\[
\begin{align*}
! (2 == 2) & \quad \text{## [1] FALSE} \\
! ((2 == 2) \lor (2 == 3)) & \quad \text{## [1] FALSE}
\end{align*}
\]
Comparing things: **Logical operators**

<table>
<thead>
<tr>
<th>And: &amp;</th>
<th>Or:</th>
<th>&amp; &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Are any of the statements FALSE&quot;?</td>
<td>&quot;Are any of the statements TRUE&quot;?</td>
<td>The &amp; operator takes precedence over</td>
</tr>
<tr>
<td>(2 == 2) &amp; (2 == 3) &amp; (4 == 4)</td>
<td>(2 == 2)</td>
<td>(2 == 3)</td>
</tr>
</tbody>
</table>
Comparing things: **Logical operators**

**Pro tip:** Use parentheses

```r
!3 == 5  # Confusing
## [1] TRUE

!(3 == 5) # Less confusing
## [1] TRUE
```
Other important points

R follows BEDMAS:

1. Brackets
2. Exponents
3. Division
4. Multiplication
5. Addition
6. Subtraction

Pro tip: Use parentheses

```r
1 + 2 * 4  # Confusing
## [1] 9

1 + (2 * 4) # Less confusing
## [1] 9
```
Your turn

Consider the following objects:

\[
\begin{aligned}
w & \leftarrow \text{TRUE} \\
x & \leftarrow \text{FALSE} \\
y & \leftarrow \text{TRUE}
\end{aligned}
\]

Write code to answer the following questions:

1. Fill in relational operators to make the following statement return \text{TRUE}:
   \[
   \neg (w \ __ \ x) \land \neg (y \ __ \ x)
   \]

2. Fill in logical operators to make this statement return \text{FALSE}:
   \[
   \neg (w \ __ \ x) \lor (y \ __ \ x)
   \]
## Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>Numbers w/decimals (aka &quot;float&quot;)</td>
<td>3.14</td>
</tr>
<tr>
<td>integer</td>
<td>Numbers w/out decimals</td>
<td>42</td>
</tr>
<tr>
<td>character</td>
<td>Text (aka &quot;string&quot;)</td>
<td>&quot;this is some text&quot;</td>
</tr>
<tr>
<td>logical</td>
<td>Used for comparing objects</td>
<td>TRUE, FALSE</td>
</tr>
</tbody>
</table>

Use `typeof()` to assess the type of any variable:

```r
typeof("hello")
```

```r
## [1] "character"
```
### Numeric types (there are 2)

<table>
<thead>
<tr>
<th>Integers</th>
<th>Doubles (aka &quot;float&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No decimals (e.g. 7)</td>
<td>Decimals (e.g. 7.0)</td>
</tr>
</tbody>
</table>
In R, numbers are "doubles" by default

Example:

```r
typeof(3)
```

### [1] "double"

Even though it *looks* like an integer, R assumes that 3 is really 3.0

Make it an integer by adding L:

```r
typeof(3L)
```

### [1] "integer"
Character types

Use single or double quotes around anything:

```r
typeof('hello')
```

```r
## [1] "character"
```

```r
typeof("3")
```

```r
## [1] "character"
```

Use single / double quotes if the string contains a quote symbol:

```r
typeof("don't")
```

```r
## [1] "character"
```
Logical types

Logical data only have two values: **TRUE** or **FALSE**

Note that these have to be in all caps, and **not** in quotes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>typeof(TRUE)</code></td>
<td><code>&quot;logical&quot;</code></td>
</tr>
<tr>
<td><code>typeof(FALSE)</code></td>
<td><code>&quot;logical&quot;</code></td>
</tr>
<tr>
<td><code>typeof('TRUE')</code></td>
<td><code>&quot;character&quot;</code></td>
</tr>
<tr>
<td><code>typeof(True)</code></td>
<td><strong>Error in typeof(True): object 'True' not found</strong></td>
</tr>
</tbody>
</table>
Logical types

Use to answer questions about logical statements.

Example: Is 1 greater than 2?

```
1 > 2
```

## [1] FALSE

Example: Is 2 greater than 1?

```
1 < 2
```

## [1] TRUE
Special values

**Infinity:** $\text{Inf}$
really big
numbers

\[
\frac{1}{0}
\]


Not a Number:
**NaN**
"not a number"

\[
\frac{0}{0}
\]

## [1] NaN

Not available: **NA**
value is "missing"

No value: **NULL**
no value
whatsoever
Your turn

Will these return **TRUE** or **FALSE**?

(try to answer first, then run the code to check)

- ! typeof('3') == typeof(3)
- (typeof(7) !== typeof("FALSE")) | FALSE
- ! (typeof(TRUE) == typeof(FALSE)) & FALSE
Final points

1) **HW 1 Preview**

⚠️ Read carefully!

2) Please take this [survey](#)