INTRODUCTION TO R PROGRAMMING

MSBA Boot Camp

DAI Lutao
Aug 30, 2019

REVIEW

REVIEW

Arithmetic Operators

Operator	Meaning	Example	Return	Python 3 Equivalence*
+	Addition	`2 + 3`	5	Base: `2 + 3` Numpy: `np.add(2,3)`
-	Subtraction	`2 - 3`	-1	Base: `2 - 3` Numpy: `np.subtract(2, 3)`
*	Multiplication	`2 * 3`	6	Base: `2 * 3` Numpy: `np.multiply(2, 3)`
/	Division	`2 / 3`	0.6666667	Base: `2 / 3` Numpy: `np.divide(2, 3)`
^	Exponential	`2 ^ 3` or `2 ** 3`	8	Base: `2 ** 3` (the only way) Numpy: `np.power(2, 3)`
%%	Modulo (remainder)	`2 %% 3`	2	Base: `2 % 3` Numpy: `np.mod(2, 3)`
%/%	Interger division	`2 %/% 3`	0	Base: `2 // 3` Numpy: `np.floor_divide(2, 3)`

REVIEW

Relational and logical Operators

Operator	Meaning	Example	Return	Python 3 Equivalence*
<	less than	`3 < 0`	FALSE	`3 < 0`
<=	less than or equal to	`3 <= 0`	FALSE	`3 <= 0`
>	greater than	`3 > 0`	TRUE	`3 > 0`
>=	greater than or equal to	`3 >= 0`	TRUE	`3 >= 0`
==	equal to	`3 == 0`	FALSE	`3 == 0`
!=	not equal to	`3 != 0`	TRUE	`3 != 0`
!x	not x	`!0` `!3`	TRUE FALSE	`not 0` `not 3`
x | y	x or y	`3 | 0`	TRUE	`True or False`
x & y	x and y	`3 & 0`	FALSE	`True and False`

In R and Python, numerical values other than 0 are treated as `TRUE`. However, `TRUE` will be converted to the integer as 1 while `FALSE` to 0 in both languages.
true and false are written as `TRUE (T)` and `FALSE (F)` in R, and `True` and `False` in Python.

REVIEW

Quiz 1

Arithmetic, relational and logical operators

- **What do the following expressions return?**

```
> 3/2
[1] 1.5

> 5 %/% 2
[1] 2

> 7%%2
[1] 1

> 3 | 0
[1] TRUE

> -2 & 5.0
[1] TRUE

> !!-9
[1] TRUE

> as.integer(TRUE)
[1] 1
```


REVIEW

Assignment operators

Name	Operator	Comment	Python 3 Equivalence*
Left assignment	<- or =	The operator <- can be used anywhere, whereas the operator = is only allowed at the top level (e.g., in the complete expression typed at the command prompt) or as one of the subexpressions in a braced list of expressions.* • In the command prompt, `x <- 5` is equivalent to `x = 5` • `mean()` is a function that returns arithmetic mean of a list of numbers. `mean(x <- 1:5)` returns 3 and defines the variable x in the environment, while `mean(x = 1:5)` returns 3 but does not define the variable x in the environment.	=
Right assignment	->	`x <- 5` is equivalent to `5 -> x`	N/A

* http://stat.ethz.ch/R-manual/R-devel/library/base/html/assignOps.html

REVIEW

Miscellaneous Operators

Operator	Description	Example	Python 3 Equivalance*

:	Create a number sequence	`> 1:4`
		`[1] 1 2 3 4`
		`> 4:1`
		`[1] 4 3 2 1`
			`>>> list(range(1, 5))`
			`[1, 2, 3, 4]`
			`>>> list(range(4, 0, -1))`
			`[4, 3, 2, 1]`
			`>>> np.arange(1, 5)`
			`array([1, 2, 3, 4])`
			`>>> np.arange(4, 0, -1)`
			`array([4, 3, 2, 1])`

%in%	Identify if an element belongs to a vector	`> 4 %in% 1:4`
		`[1] TRUE`
		`> -4 %in% 4:1`
		`[1] FALSE`
			`>>> 4 in range(1, 5)`
			`True`
			`>>> -4 in range(4, 0, -1)`
			`False`

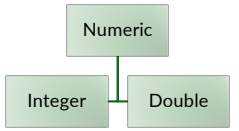
REVIEW

R Basic (atomic) classes

Atomic Classes	Example	Verify classes	Explicit Coercion
integer	`4L`	is.integer()	as.integer()
numeric (real number)	`1, 2.3, Inf`	is.numeric() is.infinite() is.double()	as.numeric() as.double()
complex	`2 + 0i`	is.complex()	as.complex()
charecter	`"I love business statistics", "3.8"`	is.character()	as.character()
logical (boolean)	`TRUE, FALSE and NA`	is.logical() is.na()	as.logical()

Check the type of variables

```
class()
> x <- pi
> class(x)
“numeric”
```



`is.numeric == TRUE` when `is.integer == TRUE` or `is.double == TRUE`

REVIEW

Workspace command

Commands	Functions
`ls()`	List all variables in the workspace
`rm(x), rm(x,y)`	Delete the variable x (and y) from the environment
`rm(list = ls())`	Remove all variables from the environment

VECTOR

CREATE A VECTOR

Vectors are sequence of data elements of the ***same*** data type (vectors are homogeneous).

```
> num_vector <- c(1, 2, 3)
> num_vector
[1] 1 2 3

> class(num_vector)
[1] "numeric"

> is.vector(num_vector)
[1] TRUE

> is.numeric(num_vector)
[1] TRUE
```

```
> new_vec <- c(1, "2", NA)
Error: unexpected input in "new_vec <- c(1, 2”

> new_vec <- c(7, NA, FALSE)
> new_vec
[1] 7 NA 0

> is.logical(c(7, NA, FALSE))
[1] FALSE

> is.logical(c(NA, FALSE))
[1] TRUE

> is.numeric(c(NA, TRUE, FALSE))
[1] FALSE
```

More on coercion later!

NA is logical

INITIALIZE A VECTOR

- **Initialize a vector – `vector()`**

```
> vector(length=5)
[1] FALSE FALSE FALSE FALSE FALSE

> vector("numeric", length=10)
 [1] 0 0 0 0 0 0 0 0 0 0
```

VECTOR COERCION

Previously

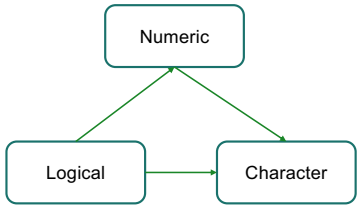
```
> is.logical(c(7, NA, FALSE))
[1] FALSE
```

How about

```
> is.numeric(c(7, NA, FALSE))
[1] TRUE
```

Now can you tell

```
> class(c("January", 2, 3, TRUE))
[1] "character"
```



Implicit coercion rules

NAME A VECTOR

Method 1 – names()

```
> price <- c(5.3, 3.5, 6.7, 11.9)
> price
[1] 5.3 3.5 6.7 11.9
> fruits <- c("apple", "orange", "banana", "mongo")
> names(price) <- fruits
> price
 apple orange banana  mongo
   5.3    3.5    6.7   11.9
```

Method 2 – Assignment

```
price <- c(apple = 5.3, orange = 3.5, banana = 6.7, mongo = 11.9)
```

Quiz – Is it syntatically correct?

```
price <- c(apple <- 5.3, orange <- 3.5,
banana <- 6.7, mongo <- 11.9)
```

If so, what's the differece?

How about this?

```
price <- c("apple" = 5.3, "orange" = 3.5,
"banana" = 6.7, "mongo" = 11.9)
```


VECTOR ARITHMETICS

Computations are performed **element-wise**

- **A vector and a single value**

```
> daily_income <- c(Tom = 10, Jimmy = 23, Lucy = 17)
> monthly_income <- daily_income * 30
> monthly_income
 Tom Jimmy Lucy
 300   690  510
```

- **A vector and a vector**

```
> monthly_expenses <- c(200, 700, 350)
> monthly_savings <- monthly_income - monthly_expenses
> monthly_savings
 Tom Jimmy Lucy
 100   -10  160
```


VECTOR ARITHMETICS

Computations are performed **element-wise**

- **Compare two vectors**

```
> c(1, 2, 3) > c(TRUE, NA, 0)
[1] FALSE NA TRUE

> c(0, 1, 2) & c(7, 8, 9)
[1] FALSE TRUE TRUE

> !c(0, 1, 2)
[1] TRUE FALSE FALSE
```

VECTOR SUBSETTING

Generate a numeric vector

```
> vec <- sample(1:100, 10, replace=FALSE)
> vec
[1] 59 94 41 35 47  3  9 82 77 22

> name <- c("a", "b", "c", "d", "e", "f", "g", "h", "i", "j")
> names(vec) <- name
> vec
 a  b  c  d  e  f  g  h  i  j
59 94 41 35 47  3  9 82 77 22
```

Index	1	2	3	4	5	6	7	8	9	10
Name	a	b	c	d	e	f	g	h	i	j
Value	59	94	41	35	47	3	9	82	77	22

VECTOR SUBSETTING

- **Subsetting by indices**

```
> vec[1]
 a
59

> vec[c(3, 4, 5)]
 c  d  e
41 35 47

> vec[-1] #all but the first element
 b  c  d  e  f  g  h  i  j
94 41 35 47  3  9 82 77 22
```

same as `vec[3:5]`, but different from `vec[c(3, 5, 4)]`!

Unary operator '-' means "exclude"

Index	1	2	3	4	5	6	7	8	9	10
Name	a	b	c	d	e	f	g	h	i	j
Value	59	94	41	35	47	3	9	82	77	22

VECTOR SUBSETTING

- **Subsetting by names**

```
> vec["a"]
 a
59

> vec[c("c", "d", "e")]
 c  d  e
41 35 47

> vec[-"a"]
Error
```

Unary operator "-" does not work on names!

Index	1	2	3	4	5	6	7	8	9	10
Name	a	b	c	d	e	f	g	h	i	j
Value	59	94	41	35	47	3	9	82	77	22

VECTOR SUBSETTING

- **Subsetting by a logical vector**
  - The logical vector and `vec` have the same length

```
> vec[c(T, F, F, F, T, T, F, T, F, F)]
 a  e  f  h
59 47  3 82
```

  - The logical vector is shorter than `vec` - **Recycling**

```
> vec[c(T, F)] #equivalent to vec[c(T, F, T, F, T, F, T, F, T, F)]
 a  c  e  g  i
59 41 47  9 77
```

  - The logical vector is longer than `vec`, and the exceeding digits contain `TRUE` – Appending NA

```
> vec[c(T, F, F, F, T, T, F, T, F, F, T, T)]
a  e  f  h <NA> <NA>
59 47 3 82 NA   NA
```

Index	1	2	3	4	5	6	7	8	9	10
Name	a	b	c	d	e	f	g	h	i	j
Value	59	94	41	35	47	3	9	82	77	22

MATRIX

CREATE MATRIX

- **Method 1 – matrix()**

```
> matrix(1:6, nrow = 2)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> matrix(1:6, nrow = 2, byrow = T)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6

> matrix(1:3, nrow = 2, ncol = 3)
     [,1] [,2] [,3]
[1,]    1    3    2
[2,]    2    1    3
```

equivalent to `matrix(1:6, ncol = 3)`

`byrow` controls the filling directions
- `byrow=F` filled by columns
- `byrow=T` filled by rows

recycling

CREATE MATRIX

- **Method 2 – `rbind()` and `cbind()`**

```
> rbind(1:3, 2:4)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    3    4

> m <- matrix(1:6, nrow = 2)
> cbind(m, 98:99)
     [,1] [,2] [,3] [,4]
[1,]    1    3    5   98
[2,]    2    4    6   99
```

`rbind()`: binding vectors or matrices as rows

`cbind()`: binding vectors or matrices as columns

NAME ROWS AND COLUMNS OF A MATRIX

- **Method 1 - rownames() and colnames()**

```
> m <- matrix(1:6, nrow = 2)
> rownames(m) <- c('row1','row2')
> colnames(m) <- c('col1', 'col2', 'col3')
> m
     col1 col2 col3
row1    1    3    5
row2    2    4    6
```

Functions
Naming rows and columns **after** defining the matrix

- **Method 2 – dimnames**

```
> n <- matrix(1:6, nrow = 2,
              dimnames = list(c('row1', 'row2'),
                              c('col1', 'col2', 'col3')))
> n
     col1 col2 col3
row1    1    3    5
row2    2    4    6
```

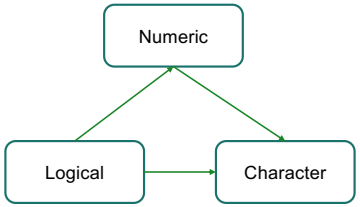
Argument
Naming rows and columns **when** defining the matrix

Naming for other R objects follows similar ways

MATRIX COERCION

```
> num <- matrix(1:6, nrow = 3)

> char <- matrix(LETTERS[1:6], nrow = 3, ncol = 4)

> cbind(num, char)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,] "1"  "4"  "A"  "D"  "A"  "D"
[2,] "2"  "5"  "B"  "E"  "B"  "E"
[3,] "3"  "6"  "C"  "F"  "C"  "F"
```

```
> LETTERS[1:6]
[1] "A" "B" "C" "D" "E" "F"
```



Same coercion rule as vectors!

MATRIX INDEXING AND SUBSETTING

```
> m <- matrix(sample(20, 12), nrow = 3)
> m
     [,1] [,2] [,3] [,4]
[1,]    1   11   20   19
[2,]   18   17   12    9
[3,]    7   14    8    5
```

How to access the value 19 (up-right corner) in the matrix?

```
> m[1,4]
[1] 19
```

row index first!

```
> m[10]
[1] 19
```

$1_{[1]}$ $11_{[4]}$ $20_{[7]}$ $19_{[10]}$
$18_{[2]}$ $17_{[5]}$ $12_{[8]}$ $9_{[11]}$
$7_{[3]}$ $14_{[6]}$ $8_{[9]}$ $5_{[12]}$

MATRIX INDEXING AND SUBSETTING

```
> m <- matrix(sample(20, 12), nrow = 3)
> m
     [,1] [,2] [,3] [,4]
[1,]    1   11   20   19
[2,]   18   17   12    9
[3,]    7   14    8    5
```

How to obtain the second row of the matrix?

```
> m[2,]
[1] 18 17 12  9

> m[2]
[1] 18
```

Mind the difference

How to obtain the second column of the matrix?

```
> m[,2]
[1] 11 17 14
```

Obtain a subset of the matrix

```
> m[1:2, 1:3]
     [,1] [,2] [,3]
[1,]    1   11   20
[2,]   18   17   12
```

Quiz
What is `m[c(1,3), 2]` and `m[,c(4,3)]` ?

MATRIX ARITHMETICS

- **Element-wise!**

```
> m
     [,1] [,2] [,3] [,4]
[1,]    1   11   20   19
[2,]   18   17   12    9
[3,]    7   14    8    5

>  m/10
     [,1] [,2] [,3] [,4]
[1,]  0.1  1.1  2.0  1.9
[2,]  1.8  1.7  1.2  0.9
[3,]  0.7  1.4  0.8  0.5

> (m / 10) + (m + 1)
     [,1] [,2] [,3] [,4]
[1,]  2.1 13.1 23.0 21.9
[2,] 20.8 19.7 14.2 10.9
[3,]  8.7 16.4  9.8  6.5
```

Sum over rows and columns – `rowSums()` and `colSums()`

```
> rowSums(m)
[1] 51 56 34

> colSums(m)
[1] 26 42 40 33
```

MATRIX ARITHMETICS

- **Multiplications**

```
> x<-matrix(c(3,4,-2,6),nrow=2,byrow=T)
> x
     [,1] [,2]
[1,]    3    4
[2,]   -2    6
```

Element-wise multiplication

```
> x * x
     [,1] [,2]
[1,]    9   16
[2,]    4   36
```

Element-wise multiplication: *
Matrix multiplication: %*%

Matrix multiplication

```
> x %*% x
     [,1] [,2]
[1,]    1   36
[2,]  -18   28
```


MATRIX FUNCTIONS

- **The dimension of a matrix – `dim()`**

```
> m
     [,1] [,2] [,3] [,4]
[1,]    1   11   20   19
[2,]   18   17   12    9
[3,]    7   14    8    5

> dim(m)
[1] 3 4
```

- **Matrix transpose – `t()`**

```
> t(m)
     [,1] [,2] [,3]
[1,]    1   18    7
[2,]   11   17   14
[3,]   20   12    8
[4,]   19    9    5
```

Inverse of a matrix – `solve()`

```
> dim(x)
[1] 2 2

> solve(x) %*% x
       [,1]          [,2]
[1,]    1 -1.110223e-16
[2,]    0  1.000000e+00
```


LIST

CREATE A LIST

- Recall that vectors can only contain elements of the **same** class

```
> c('A+', 4.3, 'A', 3.9)
[1] "A+"  "4.3" "A"   "3.9"
```

Coercion occurs!

- Lists are a special type of vector that can contain element of **different** classes

```
> grade <- list('A+', 4.3, 'A', 3.9)
> grade
[[1]]
[1] "A+"

[[2]]
[1] 4.3

[[3]]
[1] "A"

[[4]]
[1] 3.9
```

Quiz!
Is this a valid expression in R?
`list('A+', 4.3, c(4.3, '4'))`

NAME A LIST

- **Method 1 – names()**

```
> names(grade) <- c('MSBA7001','MSBA7002','MSBA7003','MSBA7004')
> grade
$MSBA7001
[1] "A+"

$MSBA7002
[1] 4.3

$MSBA7003
[1] "A"

$MSBA7004
[1] 3.9
```

As covered before, naming matrices also comes with two ways: functions and arguments

- **Method 2 – on the fly**

```
> grade <- list('MSBA7001' = 'A+', 'MSBA7002' = 4.3, 'MSBA7003' = 'A', 'MSBA7004' = 3.9)
```


LIST INDEXING AND SUBSETTING

- **Difference between [[]] and []**

```
> x <- seq(1,10,by=3)
> y <- rep(4,3)
> z <- list(x, y)
> z
[[1]]
[1]  1  4  7 10

[[2]]
[1] 4 4 4
```

In short, subsetting with [] results a list, while subsetting with [[]] results the "content" of the returning list

```
> z[2]
[[1]]
[1] 4 4 4

> z[[2]]
[1] 4 4 4

> z[2][3]
[[1]]
NULL

> z[[2]][3]
[1] 4
```

```
> class(z[2])
[1] "list"
```

```
> class(z[[2]])
[1] "numeric"
```

List

LIST INDEXING AND SUBSETTING

- **Quiz – what is the output of the following statement?**

```
> z[c(1)]
[[1]]
[1]  1  4  7 10

> z[c(1,3)]
[[1]]
[1]  1  4  7 10

[[2]]
NULL

> z[[c(1,3)]]
[1] 7
```

Recall:

```
> z
[[1]]
[1]  1  4  7 10

[[2]]
[1] 4 4 4
```

Equivalent to
`z[[1]][[3]]` or `z[[1]][3]`
get the third element of the first element

$ AND EXTENDING A LIST

- **Only works for a named list!**

```
> grade$MSBA7001
[1] "A+"

> grade$comment <- 'This student is doing very well'
> grade[['certificate']] <- 'First Class'
> str(grade)
List of 6
 $ MSBA7001   : chr "A+"
 $ MSBA7002   : num 4.3
 $ MSBA7003   : chr "A"
 $ MSBA7004   : num 3.9
 $ comment    : chr "This student is doing very well"
 $ certificate: chr "First Class"
```

To sum up, two ways of extending a list
1. assign values to `list$name`
2. assign values to `list[[name]]`

Recall

```
> grade
$MSBA7001
[1] "A+"

$MSBA7002
[1] 4.3

$MSBA7003
[1] "A"

$MSBA7004
[1] 3.9
```

LIST SUBSETTING: PARTIAL MATCHING

- **Partial matching of name is allowed for $ and [[**

```
> grade[['co']]
NULL

> grade[['co', exact=FALSE]]
[1] "This student is doing very well"

> grade$co
[1] "This student is doing very well"
```

Use with caution!

```
> grade
$MSBA7001
[1] "A+"

$MSBA7002
[1] 4.3

$MSBA7003
[1] "A"

$MSBA7004
[1] 3.9

$comment
[1] "This student is doing
very well"

$certificate
[1] "First Class"
```


CAT WITH LIST

FACTOR

CREATE FACTORS

- **Factors – Categorical variables**

```
> blood <- c('AB', 'A', 'A', 'O', 'B', 'O')
> class(blood)
[1] "character"

> blood_factor <- factor(blood)
> blood_factor
[1] AB A  A  O  B  O
Levels: A AB B O

> class(blood_factor)
[1] "factor"
```

Function `factor()` turns vectors to factors

Levels sorted alphabetically

MANUPULATING FACTORS

- **To inspect the structure of a factor - `str()`**

```
> str(blood_factor)
 Factor w/ 4 levels "A","AB","B","O": 2 1 1 4 3 4
```

A	1
AB	2
B	3
O	4

AB	2
A	1
A	1
O	4
B	3
O	4

Recall
```
> blood_factor
[1] AB A  A  O  B  O
Levels: A AB B O
```

- **Reorder levels**

```
> blood_factor <- factor(blood, levels = c('O', 'A','B', 'AB'))
> str(blood_factor)
 Factor w/ 4 levels "O","A","B","AB": 4 2 2 1 3 1
```


RENAME FACTOR LEVELS

- **Method 1 – `levels()`**

```
> levels(blood_factor) <- c('BT_O', 'BT_A', 'BT_B', 'BT_AB')
> blood_factor
[1] BT_AB BT_A  BT_A  BT_O  BT_B  BT_O
Levels: BT_O BT_A BT_B BT_AB
```

- **Method 2 – on the fly, the `labels` attribute**

```
> factor(blood, labels = c('BT_A', 'BT_AB', 'BT_B', 'BT_O'))
[1] BT_AB BT_A  BT_A  BT_O  BT_B  BT_O
Levels: BT_A BT_AB BT_B BT_O
```

Mind the order, or it will be extremely confusing!

- **A good practice – Define levels and labels explicitly**

```
> factor(blood,
+        levels = c('O', 'A', 'B', 'AB'),
+        labels = c('BT_O', 'BT_A', 'BT_B', 'BT_AB'))
[1] BT_AB BT_A  BT_A  BT_O  BT_B  BT_O
Levels: BT_O BT_A BT_B BT_AB
```


NOMINAL VERSUS ORDINAL

- Since factors, or categorical variables are merely placeholders, usually they are **nominal** and can't be numerically compared nor follow a specific order. Howerver, in some cases, **ordinal** levels are desired

```
> tshirt <- c("M", "S", "M", "L", "L", "S")
> tshirt_factor <- factor(tshirt, ordered = T,
+                         levels = c('S', 'M', 'L'))
> tshirt_factor
[1] M S M L L S
Levels: S < M < L

> tshirt_factor[2] > tshirt_factor[1]
[1] FALSE
```

`ordered = TRUE` tell `factor()` that the levels follow ascending order

DATA FRAME

CREATE A DATAFRAME

- **Use `data.frame()` to create a data frame**

```
> name <- c('Lucy', 'Helen', 'Johnny', 'Jimmy', 'Victor', 'Hilda', 'Anthony')
> age <- c(21, 23, 24, 22, 25, 21, 25)
> attendance <- c(T, F, T, F, F, T, F)

> df <- data.frame(name, age, attendance)
> df
     name age attendance
1    Lucy  21       TRUE
2   Helen  23      FALSE
3  Johnny  24       TRUE
4   Jimmy  22      FALSE
5  Victor  25      FALSE
6   Hilda  21       TRUE
7 Anthony  25      FALSE
```


EXAMINE DATA FRAMES

- **Method 1 – `str()`**

```
> str(df)
'data.frame':       7 obs. of  3 variables:
 $ name      : Factor w/ 7 levels "Anthony","Helen",..: 6 2 5 4 7 3 1
 $ age       : num  21 23 24 22 25 21 25
 $ attendance: logi  TRUE FALSE TRUE FALSE FALSE TRUE ...
```

- **Method 2 – `summary()`**

```
> summary(df)
      name        age       attendance
 Anthony:1   Min.   :21.0   Mode :logical
 Helen  :1   1st Qu.:21.5   FALSE:4
 Hilda  :1   Median :23.0   TRUE :3
 Jimmy  :1   Mean   :23.0   NA's :0
 Johnny :1   3rd Qu.:24.5
 Lucy   :1   Max.   :25.0
 Victor :1
```

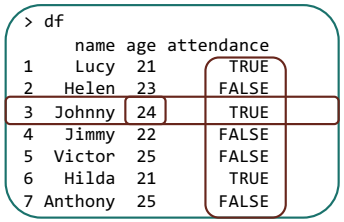

SUBSET OF A DATA FRAME

- **Data frame is essentially a list**

```
> is.data.frame(df)
[1] TRUE

> is.list(df)
[1] TRUE
```

- Therefore, **subsetting a data frame is very similar to subsetting a list**
  - Double index to access the **row** or **column** of the data frame

```
> df[3,]
    name age attendance
3 Johnny  24       TRUE

> df[,3]
[1]  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE

> df[3, 'age'] # equivalent to df[3,2]
[1] 24
```



SUBSET OF A DATA FRAME

- Single index to access the **list** that the data frame consisting of

```
> df[3]
  attendance
1       TRUE
2      FALSE
3       TRUE
4      FALSE
5      FALSE
6       TRUE
7      FALSE
```

```
> df
     name age attendance
1    Lucy  21       TRUE
2   Helen  23      FALSE
3  Johnny  24       TRUE
4   Jimmy  22      FALSE
5  Victor  25      FALSE
6   Hilda  21       TRUE
7 Anthony  25      FALSE
```

Quiz
What will `df[[3]]` return?

```
> df[3][2]
Error in `[.data.frame`(df[3], 2) : undefined columns selected

> df[[3]][2]
[1] FALSE
```

SUBSET OF A DATA FRAME

- Works for logical indexing too

Quiz
What is `df[attendance,]`?

```
> df[attendance, c('age', 'name')]
  age    name
1  21    Lucy
3  24  Johnny
6  21   Hilda
```

Pay attention to the order of columns

```
> df
      name age attendance
1     Lucy  21       TRUE
2    Helen  23      FALSE
3  Johnny  24       TRUE
4    Jimmy  22      FALSE
5   Victor  25      FALSE
6    Hilda  21       TRUE
7 Anthony  25      FALSE
```

```
> df[attendance,]
      name age attendance
1     Lucy  21       TRUE
3  Johnny  24       TRUE
6    Hilda  21       TRUE
```


EXTEND A DATA FRAME

- **Add a new column**

Method 1 - $

```
> df$height <- c(178, 189, 185, 165,
+ 176, 167, 164)
> df
      name age attendance height
1     Lucy  21       TRUE    178
2    Helen  23      FALSE    189
3  Johnny  24       TRUE    185
4    Jimmy  22      FALSE    165
5   Victor  25      FALSE    176
6    Hilda  21       TRUE    167
7 Anthony  25      FALSE    164
```

Method 2 – `cbind()`

```
> weight <- c(78, 89, 85, 65, 76, 67, 64)

> cbind(df, weight)
      name age attendance height weight
1     Lucy  21       TRUE    178     78
2    Helen  23      FALSE    189     89
3  Johnny  24       TRUE    185     85
4    Jimmy  22      FALSE    165     65
5   Victor  25      FALSE    176     76
6    Hilda  21       TRUE    167     67
7 Anthony  25      FALSE    164     64
```

Quiz
What is `df` after executing
`> cbind(df, weight)`

EXTEND A DATA FRAME

- **Add a new row – rbind()**

```
> gary <- data.frame("gary", 21, T, 183, 83)
> rbind(df, gary)
Error in match.names(clabs, names(xi)) :
  names do not match previous names

> colnames(gary)
[1] "X.gary." "X21"     "T"       "X183"    "X83"

> colnames(gary) <- c('name', 'age', 'attendance', 'height', 'weight')
> rbind(df, gary)
     name age attendance height weight
1    Lucy  21       TRUE    178     78
2   Helen  23      FALSE    189     89
3  Johnny  24       TRUE    185     85
4   Jimmy  22      FALSE    165     65
5  Victor  25      FALSE    176     76
6   Hilda  21       TRUE    167     67
7 Anthony  25      FALSE    164     64
8    gary  21       TRUE    183     83
```

`rbind()` is the only way to append a new row to a data frame

Accessing column names:
`names(df)` for data frame and `colnames(df)` for matrix.
Accessing row names:
`row.names(df)` for data frame and `rownames(df)` for matrix.

SORT DATA FRAMES

```
> sort(df$age)
[1] 21 21 22 23 24 25 25

> df$age
[1] 21 23 24 22 25 21 25
```

Sorting does not occur inplace

Quiz!
Will this work?
`> df$age <- sort(df$age)`

```
> rank <- order(df$age)
> rank
[1] 1 6 4 2 3 5 7
```

```
> df
      name age attendance height weight
1     Lucy  21       TRUE    178     78
2    Helen  23      FALSE    189     89
3  Johnny  24       TRUE    185     85
4    Jimmy  22      FALSE    165     65
5   Victor  25      FALSE    176     76
6    Hilda  21       TRUE    167     67
7 Anthony  25      FALSE    164     64
```

```
> df[rank,]
      name age attendance height weight
1     Lucy  21       TRUE    178     78
6    Hilda  21       TRUE    167     67
4    Jimmy  22      FALSE    165     65
2    Helen  23      FALSE    189     89
3  Johnny  24       TRUE    185     85
5   Victor  25      FALSE    176     76
7 Anthony  25      FALSE    164     64
```

SORT DATA FRAMES

- **To sum up**

```
df[order(df$age),]
```

- **To sort it in the descending order**

```
> df[order(df$age, decreasing = T),]
     name age attendance height weight
5  Victor  25      FALSE    176     76
7 Anthony  25      FALSE    164     64
3  Johnny  24       TRUE    185     85
2   Helen  23      FALSE    189     89
4   Jimmy  22      FALSE    165     65
1    Lucy  21       TRUE    178     78
6   Hilda  21       TRUE    167     67
```

descending is the argument of `order()`

DPYLR

CONVERT A DATA FRAME TO A DATA MATRIX

- **as.matrix() vs data.matrix()**

```
> as.matrix(df)
     name      age  attendance height weight
[1,] "Lucy"    "21" " TRUE"    "178"  "78"
[2,] "Helen"   "23" "FALSE"    "189"  "89"
[3,] "Johnny"  "24" " TRUE"    "185"  "85"
[4,] "Jimmy"   "22" "FALSE"    "165"  "65"
[5,] "Victor"  "25" "FALSE"    "176"  "76"
[6,] "Hilda"   "21" " TRUE"    "167"  "67"
[7,] "Anthony" "25" "FALSE"    "164"  "64"

> data.matrix(df)
     name age attendance height weight
[1,]    6  21          1    178     78
[2,]    2  23          0    189     89
[3,]    5  24          1    185     85
[4,]    4  22          0    165     65
[5,]    7  25          0    176     76
[6,]    3  21          1    167     67
[7,]    1  25          0    164     64
```

`as.*()` follows the coercion rule we discussed before

`data.matrix()` will try to express everything with numerical values

I bet almost always, you want to use `data.matrix()` over `as.matrix()`

THE FLIGHTS DATA SET

```
> str(flights)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame': 336776 obs. of  19 variables:
 $ year          : int  2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
 $ month         : int  1 1 1 1 1 1 1 1 1 1 ...
 $ day           : int  1 1 1 1 1 1 1 1 1 1 ...
 $ dep_time      : int  517 533 542 5
 $ sched_dep_time: int  515 529 540 5
 $ dep_delay     : num  2 4 2 -1 -6 -4 -5 -3 -2 ...
 $ arr_time      : int  830 850 923 1004 812 740 913 709 838 753 ...
 $ sched_arr_time: int  819 830 850 1022 837 728 854 723 846 745 ...
 $ arr_delay     : num  11 20 33 -18 -25 12 19 -14 -8 8 ...
 $ carrier       : chr  "UA" "UA" "AA
 $ flight        : int  1545 1714 114
 $ tailnum       : chr  "N14228" "N24
 $ origin        : chr  "EWR" "LGA"
 $ dest          : chr  "IAH" "IAH" "MIA" "BQN" ...
 $ air_time      : num  227 227 160 183 116 150 158 53 140 138 ...
 $ distance      : num  1400 1416 1089 1576 762 ...
 $ hour          : num  5 5 5 5 6 5 6 6 6 6 ...
 $ minute        : num  15 29 40 45 0 58 0 0 0 0 ...
 $ time_hour     : POSIXct, format: "2013-01-01 05:00:00" "2013-01-01 05:00:00"
"2013-01-01 05:00:00" ...
```

flights is a data frame!

```
> class(flights)
[1] "tbl_df"     "tbl"        "data.frame"
```

To know what each variable represent, please read the documentation

```
?flights
```

PIPE (%>%)

- **To execute commands in sequence**

```
data.frame %>%
  command 1 %>%
    command 2 %>%
      command 3 %>%
        …
```

dplyr commands

dplyr Command	SQL equivalent	Action
filter()	WHERE	Filter df based on a set of condition
select()	SELECT	Choose only certain variables (columns)
distinct()	DISTINCT	De-duplicate result-set
arrange()	ORDER BY	Order results
rename()	SELECT	Rename variables
mutate()	SELECT	Create new variables
group_by()	GROUP BY	Group rows
summarise()	SELECT	Create new variable in grouped setting

DPLYR::FILTER()

```
> flights %>%
+     filter(month == 1 & day == 1)

# A tibble: 842 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     1      517            515         2      830
 2  2013     1     1      533            5
 3  2013     1     1      542            5
 4  2013     1     1      544            5
 5  2013     1     1      554            6
 6  2013     1     1      554            5
 7  2013     1     1      555            6
 8  2013     1     1      557            6
 9  2013     1     1      557            6
10  2013     1     1      558            6
# ... with 832 more rows, and 12 more variables:
#   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

The operation does not occur in-place, so don't forget to assign the returned sub-dataframe to a new variable!
I will not repeat the idea in the following slides.

```
> filteredFlight <- flights %>%
+     filter(month == 1 & day == 1)
```


DPLYR::SELECT

```
> flights %>%
+     filter(dest %in% c("PHL","SLC") & month <= 6) %>%
+     select(origin, dest, carrier)

# A tibble: 2,116 x 3
   origin  dest carrier
    <chr> <chr>   <chr>
 1    JFK   SLC      DL
 2    LGA   PHL      US
 3    JFK   SLC      DL
 4    JFK   SLC      DL
 5    EWR   SLC      DL
 6    JFK   SLC      DL
 7    JFK   PHL      9E
 8    JFK   SLC      B6
 9    JFK   SLC      DL
10    JFK   PHL      9E
# ... with 2,106 more rows
```

Quiz
What if we want to exclude some variables?

DPLYR::SELECT

```
> flights %>%
+     filter(dest %in% c("PHL","SLC") & month <= 6) %>%
+     select(-year, -month, -day)

# A tibble: 2,116 x 16
   dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
      <int>          <int>     <dbl>    <int>          <int>     <dbl>   <chr>
 1      655            655         0     1021           1030        -9      DL
 2      908            915        -7     1004           1033       -29      US
 3     1047           1050        -3     1401           1410        -9      DL
 4     1245           1245         0     1616           1615         1      DL
 5     1323           1300        23     1651           1608        43      DL
 6     1543           1550        -7     1933           1925         8      DL
 7     1600           1610       -10     1712           1729       -17      9E
 8     1909           1912        -3     2239           2237         2      B6
 9     1915           1920        -5     2238           2257       -19      DL
10     2000           2000         0     2054           2110       -16      9E
# ... with 2,106 more rows, and 9 more variables: flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
```

DPLYR::DISTINCT

```
> flights %>%
+     filter(dest %in% c("PHL","SLC") & month <= 6) %>%
+     select(origin, dest, carrier) %>%
+     distinct()

# A tibble: 8 x 3
  origin  dest carrier
   <chr> <chr>   <chr>
1    JFK   SLC      DL
2    LGA   PHL      US
3    EWR   SLC      DL
4    JFK   PHL      9E
5    JFK   SLC      B6
6    EWR   PHL      EV
7    JFK   PHL      US
8    JFK   PHL      DL
```

distinct() removes all duplicate rows

DPLYR::ARRANGE

```
> flights %>%
+     filter(dest %in% c("PHL","SLC") & month <= 6) %>%
+     select(origin, dest, carrier) %>%
+     distinct() %>%
+     arrange(origin, desc(dest), carrier)

# A tibble: 8 x 3
  origin  dest carrier
   <chr> <chr>   <chr>
1    EWR   SLC      DL
2    EWR   PHL      EV
3    JFK   SLC      B6
4    JFK   SLC      DL
5    JFK   PHL      9E
6    JFK   PHL      DL
7    JFK   PHL      US
8    LGA   PHL      US
```

Arrange() sorts the data frame variables in alphabetical order by default, and in descending order if explicitly specified.

In this case, the data frame is first sorted alphabetically by origin, and then in descending order by dest, and finally, in ascending order by carrier.

DPLYR::RENAME

```
> flights %>%
+     filter(dest %in% c("PHL","SLC") & month <= 6) %>%
+     select(origin, dest, carrier) %>%
+     distinct() %>%
+     arrange(origin, desc(dest), carrier) %>%
+     rename(airline = carrier)

# A tibble: 8 x 3
  origin  dest airline
   <chr> <chr>   <chr>
1    EWR   SLC      DL
2    EWR   PHL      EV
3    JFK   SLC      B6
4    JFK   SLC      DL
5    JFK   PHL      9E
6    JFK   PHL      DL
7    JFK   PHL      US
8    LGA   PHL      US
```

rename(new_name = old_name)

In some package, the renaming rule can be old_name = new_name. Please bear this in mind.

DPYLR::MUTATE

```
> flights %>%
+     mutate(gain          = dep_delay - arr_delay,
+            speed         = distance / air_time * 60,
+            gain_per_hour = gain / (air_time / 60)) %>%
+     select(dep_delay, arr_delay, gain, distance, distance, air_time, speed, gain_per_hour)

# A tibble: 336,776 x 7
   dep_delay arr_delay  gain distance air_time    speed gain_per_hour
       <dbl>     <dbl> <dbl>    <dbl>    <dbl>    <dbl>         <dbl>
 1         2        11    -9     1400      227 370.0441     -2.378855
 2         4        20   -16     1416      227 374.2731     -4.229075
 3         2        33   -31     1089      160 408.3750    -11.625000
 4        -1       -18    17     1576      183 516.7213      5.573770
 5        -6       -25    19      762      116 394.1379      9.827586
 6        -4        12   -16      719      150 287.6000     -6.400000
 7        -5        19   -24     1065      158 404.4304     -9.113924
 8        -3       -14    11      229       53 259.2453     12.452830
 9        -3        -8     5      944      140 404.5714      2.142857
10        -2         8   -10      733      138 318.6957     -4.347826
# ... with 336,766 more rows
```

DPLYR::GROUP_BY & DPLYR::SUMMARISE

```
> flights %>%
+     group_by(origin) %>%
+     summarise(num_of_flights = n(),
+               avg_delay      = mean(dep_delay, na.rm = TRUE))
# A tibble: 3 x 3
  origin num_of_flights avg_delay
   <chr>          <int>     <dbl>
1    EWR         120835  15.10795
2    JFK         111279  12.11216
3    LGA         104662  10.34688
```

group_by() usually goes with summarise() – The reason why you want to group entries is usually to get the group statistics!

Common functions or arguments in dplyr::summarise()

dplyr::n() The number of observations in the current group

dplyr::n_distinct() Efficiently count the number of unique values in a set of vector

na.rm = TRUE Removing NA before performing calculation

DPLYR::GROUP_BY & DPLYR::SUMMARISE

```
> flights %>%
+     filter(!is.na(dep_delay) & !is.na(arr_delay)) %>%
+     group_by(dep_delay > 0, arr_delay > 0) %>%
+     summarise(num_of_flights = n())

# A tibble: 4 x 3
# Groups:   dep_delay > 0 [?]
  `dep_delay > 0` `arr_delay > 0` num_of_flights
            <lgl>           <lgl>          <int>
1           FALSE           FALSE         158900
2           FALSE            TRUE          40701
3            TRUE           FALSE          35442
4            TRUE            TRUE          92303
```

group_by can also take expression

Quiz
What does 35442 mean?

EXPLAIN THE ANALYSIS

```
> flights %>%
+     group_by(origin) %>%
+     summarise(destinations = n_distinct(dest),
+               avg_distance = mean(distance, na.rm = TRUE))


# A tibble: 3 x 3
  origin destinations avg_distance
   <chr>        <int>        <dbl>
1    EWR           86    1056.7428
2    JFK           70    1266.2491
3    LGA           68     779.8357
```


EXPLAIN THE ANALYSIS

```
> flights %>%
+     summarise(num_of_flights = n(),
+               destinations   = n_distinct(dest),
+               avg_delay      = mean(dep_delay, na.rm = TRUE))


# A tibble: 1 x 3
  num_of_flights destinations avg_delay
           <int>        <int>     <dbl>
1         336776          105  12.63907
```

EXPLAIN THE ANALYSIS

```
> flights %>%
+     group_by(origin, dest) %>%
+     summarise(max_distance = max(distance)) %>%
+     arrange(desc(max_distance))

# A tibble: 224 x 3
# Groups:   origin [3]
   origin  dest max_distance
    <chr> <chr>        <dbl>
 1    JFK   HNL         4983
 2    EWR   HNL         4963
 3    EWR   ANC         3370
 4    JFK   SFO         2586
 5    JFK   OAK         2576
 6    JFK   SJC         2569
 7    EWR   SFO         2565
 8    JFK   SMF         2521
 9    JFK   LAX         2475
10    JFK   BUR         2465
# ... with 214 more rows
```


COMMON DATA ANALYSIS OPERATION EXAMPLES

REMOVING MISSING VALUES (NA) VALUES

- **Removing missing values in vectors**

```
> vec <- c(1, 4, NA, 9, NA, 2)
> vec
[1]  1  4 NA  9 NA  2
> bad <- is.na(vec)
> vec[!bad]
[1] 1 4 9 2
```

The key idea is to first get the indices of missing/non-missing values, and then apply subsetting

- **Removing missing values in data frames**

```
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

```
> good <- complete.cases(airquality)
> head(airquality[good,])
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
7    23     299  8.6   65     5   7
8    19      99 13.8   59     5   8
```


PLOTTING

GGPLOT COMMANDS

Data	Plots	Geom (ggplot command)
One Continuous	Histogram	`geom_histogram`
One Continuous + One Categorical	Boxplot	`geom_boxplot`
Two Continuous	Scatter Plot	`geom_point`
Three Continuous	Scatter Plot + Size	`geom_point w/ size aesthetic`
Two Continuous + One Categorical	Scatter Plot + Color	`geom_point w/ color aesthetic`
Categorical with reasonable number of levels	Faceting!!	`facet_wrap()`

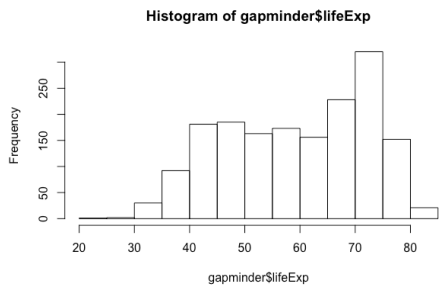
GAPMINDER DATASET

```
> summary(gapminder)
        country        continent        year         lifeExp          pop
 Afghanistan:  12   Africa  :624   Min.   :1952   Min.   :23.60   Min.   :6.001e+04
 Albania    :  12   Americas:300   1st Qu.:1966   1st Qu.:48.20   1st Qu.:2.794e+06
 Algeria    :  12   Asia    :396   Median :1980   Median :60.71   Median :7.024e+06
 Angola     :  12   Europe  :360   Mean   :1980   Mean   :59.47   Mean   :2.960e+07
 Argentina  :  12   Oceania : 24   3rd Qu.:1993   3rd Qu.:70.85   3rd Qu.:1.959e+07
 Australia  :  12                  Max.   :2007   Max.   :82.60   Max.   :1.319e+09
 (Other)    :1632
   gdpPercap
 Min.   :   241.2
 1st Qu.:  1202.1
 Median :  3531.8
 Mean   :  7215.3
 3rd Qu.:  9325.5
 Max.   :113523.1
```

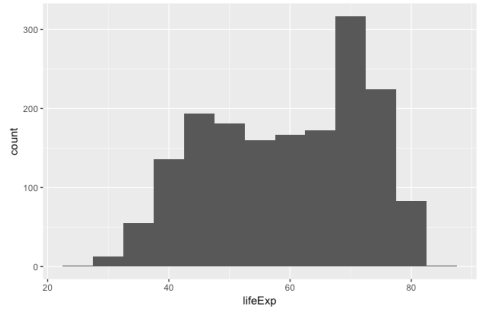
`country`	factor with 142 levels
`continent`	factor with 5 levels
`year`	ranges from 1952 to 2007 in increments of 5 years
`lifeExp`	life expectancy at birth, in years
`pop`	population
`gapPercap`	GDP per capita

ONE CONTINUOUS VARIABLE

```
> hist(gapminder$lifeExp)
```

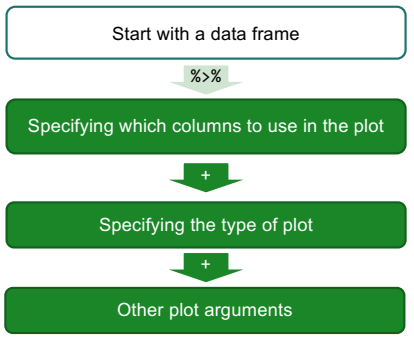



```
> gapminder %>%
+     ggplot(aes(x = lifeExp)) +
geom_histogram(binwidth = 5)
```

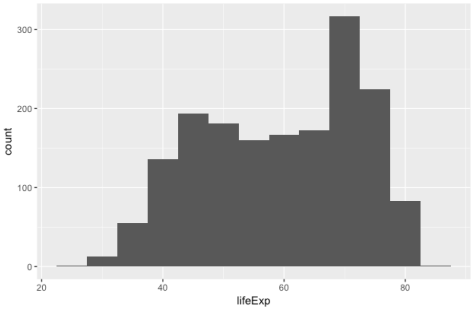


ONE CONTINUOUS VARIABLE

GGPLOT COMMAND FORMULA



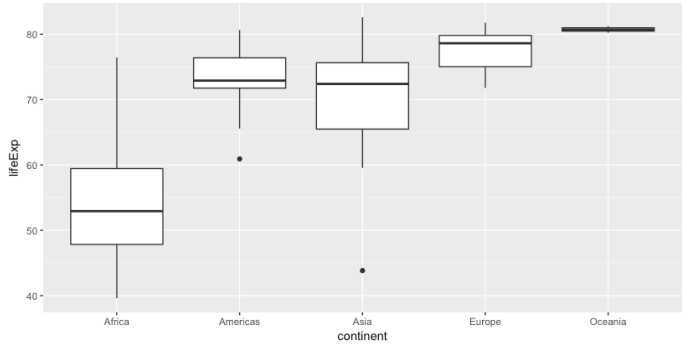

```
> gapminder %>%
+     ggplot(aes(x = lifeExp)) +
geom_histogram(binwidth = 5)
```



ONE CONTINUOUS VARIABLE + ONE CATEGORICAL VARIABLE

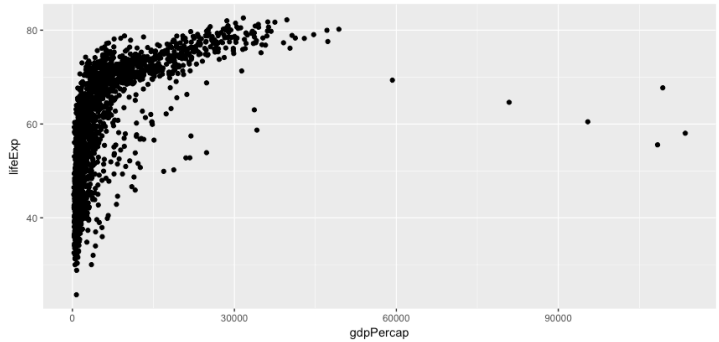
```
> gapminder %>%
+     filter(year == 2007) %>%
+     ggplot(aes(x = continent, y = lifeExp)) + geom_boxplot()
```

dpylr



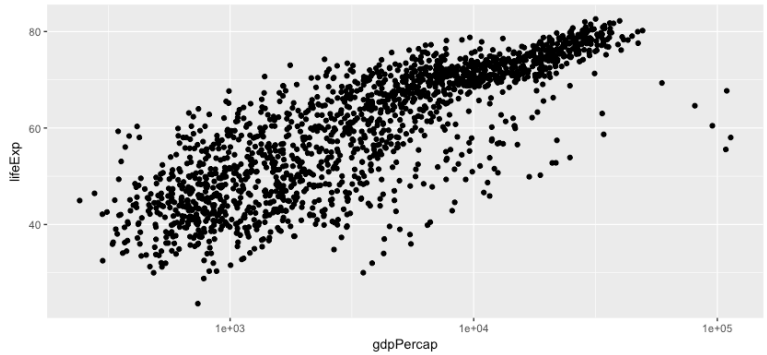
TWO CONTINUOUS VARIABLES

```
> gapminder %>%
+     ggplot(aes(x = gdpPercap, y = lifeExp)) +
+     geom_point()
```



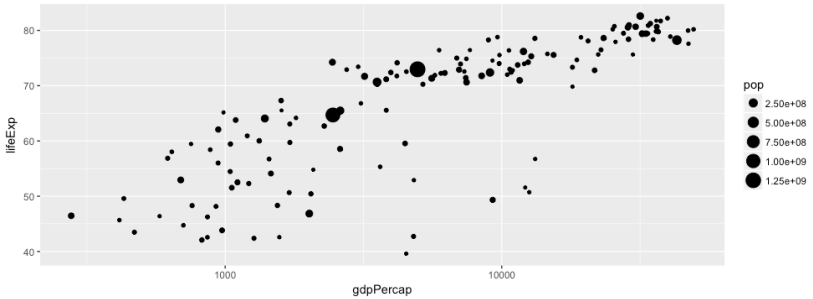
TWO CONTINUOUS VARIABLES

```
> gapminder %>%
+     ggplot(aes(x = gdpPercap, y = lifeExp)) +
+     geom_point() +
+     scale_x_log10()
```



THREE CONTINUOUS VARIABLES

```
> gapminder %>%
+     filter(year == 2007) %>%
+     ggplot(aes(x = gdpPercap, y = lifeExp, size = pop)) +
+     geom_point() +
+     scale_x_log10()
```

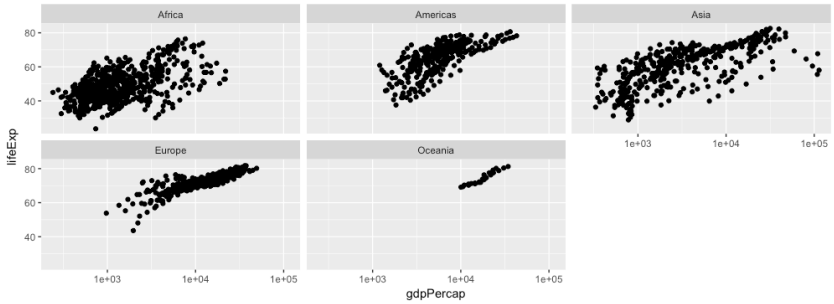


TWO CONTINUOUS VARIABLES + ONE CATEGORICAL VARIABLE (I)

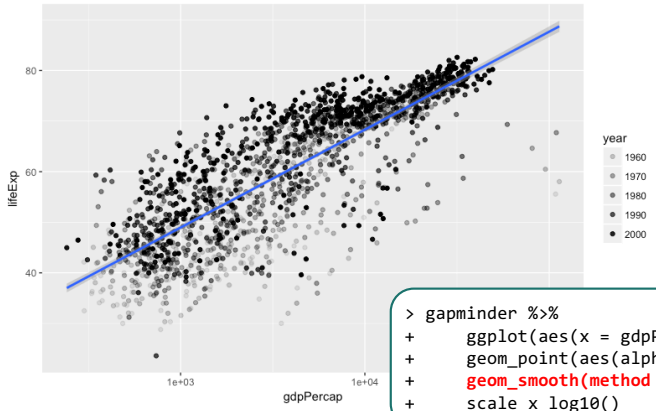
```
> gapminder %>%
+     ggplot(aes(x = gdpPercap, y = lifeExp, color = continent)) +
+     geom_point() +
+     scale_x_log10()
```

TWO CONTINUOUS VARIABLES + ONE CATEGORICAL VARIABLE (II)

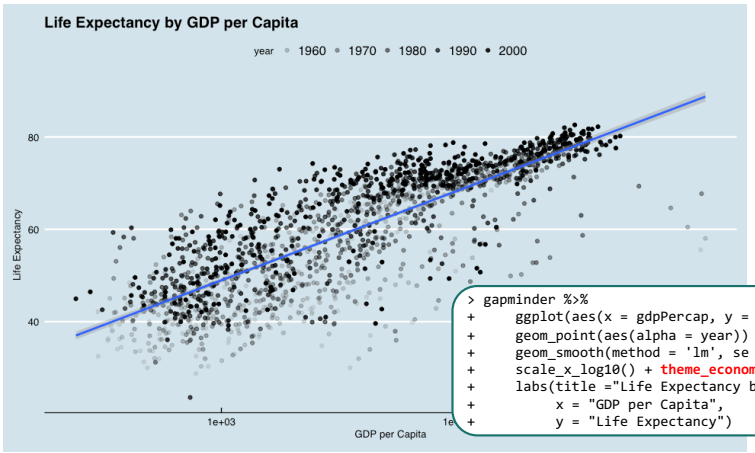
```
> gapminder %>%
+     ggplot(aes(x = gdpPercap, y = lifeExp)) +
+     geom_point() +
+     scale_x_log10() +
+     facet_wrap(~continent)
```



GGPLOT + LINEAR REGRESSION




```
> gapminder %>%
+     ggplot(aes(x = gdpPercap, y = lifeExp)) +
+     geom_point(aes(alpha = year)) +
+     geom_smooth(method = 'lm', se = TRUE) +
+     scale_x_log10()
```


```
> gapminder %>%
+     ggplot(aes(x = gdpPercap, y = lifeExp)) +
+     geom_point(aes(alpha = year)) +
+     geom_smooth(method = 'lm', se = TRUE) +
+     scale_x_log10() + theme_economist() +
+     labs(title ="Life Expectancy by GDP per Capita",
+          x = "GDP per Capita",
+          y = "Life Expectancy")
```

FUNCTION AND CONTROL STRUCTURE

DEFINE A FUNCTION

```
> sum.of.squares <- function(x,y) {
+     x^2 + y^2
+ }

> sum.of.squares(3,4)
[1] 25

> degree2rad <- function(angle){
+     angle/180*pi
+ }

> sin(30)
[1] -0.9880316

> sin(degree2rad(30))
[1] 0.5
```

```
function.name <- function(arguments){
       ## do something about arguments
}
```


IF-ELSE

```
> grading <- function(score){
+     if(score < 60){
+         print('Failed')
+     } else if(score < 80){
+         print('Passed')
+     } else{
+         print('Excellent')
+     }
+ }

> grading(sample(1:100, 1))
[1] "Failed"
> grading(sample(1:100, 1))
[1] "Passed"
> grading(sample(1:100, 1))
[1] "Failed"
> grading(sample(1:100, 1))
[1] "Excellent"
```

```
if(<condition 1>) {
     ## do something
} else if(<condition 2>){
     ## do something different
} else {
     ## do something different
}
```


FOR LOOP

```
> print.grades <- function(scores){
+     for(score in scores){
+         grading(score)
+     }
+ }

> scores <- sample(1:100, 5, replace = TRUE)
> scores
[1] 32 54 87 14 62

> print.grades(scores)
[1] "Failed"
[1] "Failed"
[1] "Excellent"
[1] "Failed"
[1] "Passed"
```

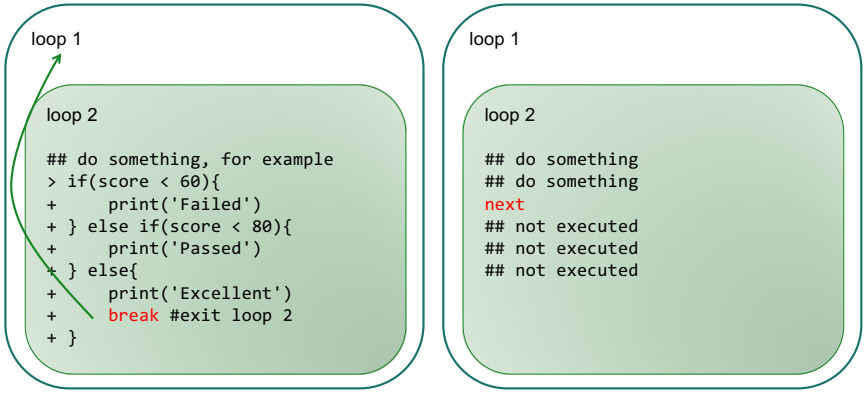
```
for(element in list/vector){
       ## do something with the element
}
```

WHILE

```
> count <- 0
> while(count < 5){
+     print(count)
+     count <- count + 1
+ }
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
```

```
while(condition){
       ## do something
       ## if the condition is TRUE
}
## do something
## if the condition is FALSE
```


BREAK AND NEXT



ADDITIONAL QUESTIONS

- **Email:**
  - lutaodai@hku.hk
- **Office:**
  - KKL 723