Tricks in R

Some clean up work using tidyverse

```
suppressWarnings(suppressMessages(library(tidyverse)))
library(tidyverse)

wage.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Wage.csv",
                     header=T, stringsAsFactors = F, na.strings = "?")
wage.df.original = tibble(wage.df)
wage.df = tibble(wage.df)


# convers maritl into an ordered factor
# find levels in maritl column
table(wage.df$maritl)[["1. Never Married"]]
```

```
## [1] 648
```

```
marit_levels <- names(table(wage.df$maritl))
typeof(marit_levels)
```

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

```
# now convert it to order factor

# iris %>% select(starts_with(c("Petal", "Sepal")))

# iris %>%
#   group_by(Species) %>%
#   summarise(across(starts_with("Sepal"), mean))

# iris %>% summarise_at("Petal.Width", funs(min, anyNA))
# starwars %>% mutate_if(is.numeric, scale2, na.rm = TRUE)

wage.df %>%
  mutate_at("maritl", ~factor(.x,levels=(.x %>% table() %>% dimnames())))
```

```
## # A tibble: 3,000 x 12
##     year   age sex   maritl race  education region jobclass health health_ins
##    <int> <int> <chr> <fct>  <chr> <chr>     <chr>  <chr>    <chr>  <chr>
##  1  2006    18 1. M~ <NA>   1. W~ 1. < HS ~ 2. Mi~ 1. Indu~ 1. <=~ 2. No
##  2  2004    24 1. M~ <NA>   1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 2. No
##  3  2003    45 1. M~ <NA>   1. W~ 3. Some ~ 2. Mi~ 1. Indu~ 1. <=~ 1. Yes
##  4  2003    43 1. M~ <NA>   3. A~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  5  2005    50 1. M~ <NA>   1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  6  2008    54 1. M~ <NA>   1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  7  2009    44 1. M~ <NA>   4. O~ 3. Some ~ 2. Mi~ 1. Indu~ 2. >=~ 1. Yes
##  8  2008    30 1. M~ <NA>   3. A~ 3. Some ~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
```

```
##  9  2006    41 1. M~ <NA>   2. B~ 3. Some ~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## 10  2004    52 1. M~ <NA>   1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## # ... with 2,990 more rows, and 2 more variables: logwage <dbl>, wage <dbl>
```

```
turn.to.factor <- function(col, na.rm = FALSE) factor(col,levels=(col %>% table() %>% dimnames()))

wage.df %>%
  mutate_if(is.character, turn.to.factor, na.rm = TRUE)
```

```
## # A tibble: 3,000 x 12
##     year   age sex   maritl race  education region jobclass health health_ins
##    <int> <int> <fct> <fct>  <fct> <fct>     <fct>  <fct>    <fct>  <fct>
##  1  2006    18 1. M~ <NA>   <NA>  <NA>      2. Mi~ <NA>     <NA>   <NA>
##  2  2004    24 1. M~ <NA>   <NA>  <NA>      2. Mi~ <NA>     <NA>   <NA>
##  3  2003    45 1. M~ <NA>   <NA>  <NA>      2. Mi~ <NA>     <NA>   <NA>
##  4  2003    43 1. M~ <NA>   <NA>  <NA>      2. Mi~ <NA>     <NA>   <NA>
##  5  2005    50 1. M~ <NA>   <NA>  <NA>      2. Mi~ <NA>     <NA>   <NA>
##  6  2008    54 1. M~ <NA>   <NA>  <NA>      2. Mi~ <NA>     <NA>   <NA>
##  7  2009    44 1. M~ <NA>   <NA>  <NA>      2. Mi~ <NA>     <NA>   <NA>
##  8  2008    30 1. M~ <NA>   <NA>  <NA>      2. Mi~ <NA>     <NA>   <NA>
##  9  2006    41 1. M~ <NA>   <NA>  <NA>      2. Mi~ <NA>     <NA>   <NA>
## 10  2004    52 1. M~ <NA>   <NA>  <NA>      2. Mi~ <NA>     <NA>   <NA>
## # ... with 2,990 more rows, and 2 more variables: logwage <dbl>, wage <dbl>
```

table and factor

```
# ------ table is used for counting records on one ore more factor columns -----
(x <- table(mtcars[c("vs", "cyl", "am")]))
```

```
## , , am = 0
##
##    cyl
## vs   4  6  8
##   0  0  0 12
##   1  3  4  0
##
## , , am = 1
##
##    cyl
## vs   4  6  8
##   0  1  3  2
##   1  7  0  0
```

```
# ---- for classification  use ordered factor to assign 1 to quality of interest --------#
x<-factor(c("a","b","b","a"))
y<- ordered(c("a","b","b","a"), levels = c("c","b","a"))
str(attributes(y))
```

```
## List of 2
##  $ levels: chr [1:3] "c" "b" "a"
##  $ class : chr [1:2] "ordered" "factor"
```

```
# -------------- modify levels of a factor -------

x <- ordered(c("a", "b", "a", "b"), levels = c("a", "b", "c"))
```

```r
table(x)
```

```
## x
## a b c
## 2 2 0
```

```r
levels(x) <- rev(levels(x))
table(x)
```

```
## x
## c b a
## 2 2 0
```

Modify in place

- Modifying an R object usually creates a new copy.
  - Use base::tracemem(x) to see if the object is copied

```r
#--------------------------------------------------------------------
# object with a single binding (object has a single name bound to it)
#--------------------------------------------------------------------

(v <- c(1,1,1))
```

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

```r
# use base::tracemem(x) to see if the object is copied
cat(tracemem(v), "\n") # start tracing reference to v
```

```
## <0x7fe2c53548c8>
```

```r
v[[3]] <- 4
```

```
## tracemem[0x7fe2c53548c8 -> 0x7fe2c62d5378]: eval eval withVisible withCallingHandlers handle timing_
```

```r
v
```

```
## [1] 1 1 4
```

```r
untracemem(v) # stop tracing reference to v
```

Atomic Vector Subsettings and Factor Subsettings

- Key Pints:
  - avoid using: *is.atomic()*, *is.numeric()* and *is.vector()*, use *is.null(dim())*
  - applying *typeof()* to a vector returns type of its elements
  - use *[ ]* to retrieve multiple values from a vector
    - using *[[ ]]* to get multiple values from a vector causes Error
  - recomanded to use *[[ ]]* to retrieve a single element from a vector
    - *[[ ]]* with zero length, NULL and out of bound index or name return error:
      - vector[[NULL]] ==> Error
      - vector[[integer(0)]] ==> Error
      - vector[[out_of_bound_integer]] ==> Error
      - vector[[out_of_bound_character]] ==> Error
    - *[ ]* with zero length, NULL and out of bound index or name return empty vector
  - only logical vector is recycled when used for subsetting

- *NA* has logical type thus when used for subsetting it will be recycled
- coercion used by *c()* is: *logical => integer => double => character*
- avoid automatic coercion inside *c()*, always do explicit coercion within *c()*
- To create an empty vector of certain "type" and certain "length"
  - vector("type", length = n) as: *vector("complex", length = 0)*
  - type(n) as: *numeric(3)*
  - type() is shorthand for type(0) as: *numeric()*

```
#-----------------------------------------------
# --- This is how R show an unnamed vector ---
#-----------------------------------------------

(x <- c(2.1, 4.2, 5.3, 1.4))
```

```
## [1] 2.1 4.2 5.3 1.4
```

```
names(x) <- c("one", "two", "three", "four")

#-----------------------------------------------
# --- This is how R show a named vector ---
#-----------------------------------------------

x
```

```
##   one   two three  four
##   2.1   4.2   5.3   1.4
```

```
typeof(x)
```

```
## [1] "double"
```

```
#-----------------------------------------------
# -------- subsetting by position (index) -----
#-----------------------------------------------
x[c(3,1)]
```

```
## three   one
##   5.3   2.1
```

```
# x[[c(3,1)]]  error: attempt to select more than one element vectorIndex

#-------------------------------------------------
# ---- duplicate index means duplicate values -----
#-------------------------------------------------
x[c(3,3,2,4,2,3,1,4)]
```

```
## three three   two  four   two three   one  four
##   5.3   5.3   4.2   1.4   4.2   5.3   2.1   1.4
```

```
# x[[c(3,3,2,4,2,3,1,4)]] error: attempt to select more than one element vectorIndex
#-------------------------------------------------------------------------------
# --- order() returns indices such that the corresponding elements are ordered--
#-------------------------------------------------------------------------------
x[order(x)]
```

```
##  four   one   two three
##   1.4   2.1   4.2   5.3
```

```r
# x[[order(x)]] error: attempt to select more than one element vectorIndex
#---------------------------------------------------------------
# --- real numbers used as indices are truncated to integers ----
#---------------------------------------------------------------
x[c(3.1, 3.2, 3.3, 4.2)]
```

```
## three three three  four
##   5.3   5.3   5.3   1.4
```

```r
# x[[c(3.1, 3.2, 3.3, 4.2)]] error: attempt to select more than one element vectorIndex
#---------------------------------------------------------------
# use negative elements to exclude values at specified position
# ---------------------------------------------------------------
x[-2]
```

```
##   one three  four
##   2.1   5.3   1.4
```

```r
# x[[-2]] error: attempt to select more than one element vectorIndex

x[c(-3:-1)]
```

```
## four
##  1.4
```

```r
# x[[c(-3:-1)]] error: attempt to select more than one element vectorIndex
#-------------------------------------------------------------------------
# Logical vectors as index makes R chooses elements with TRUE as index
#-------------------------------------------------------------------------

x[c(T,F,F,T)]
```

```
##  one four
##  2.1  1.4
```

```r
# x[[c(T,F,F,T)]] error: attempt to select more than one element vectorIndex

x[x>3]
```

```
##   two three
##   4.2   5.3
```

```r
# x[[x>3]] error: attempt to select more than one element vectorIndex
#--------------------------------
# Only logical index is recycled
#--------------------------------
x[T]
```

```
##   one   two three  four
##   2.1   4.2   5.3   1.4
```

```r
x[[T]] # recomended
```

```
## [1] 2.1
```

```r
#--------------------------------
# NA in index causes NA in output
#--------------------------------
x[c(2,3,NA,4)]
```

```
##   two three  <NA>  four
##   4.2   5.3    NA   1.4
```

```
# x[[c(2,3,NA,4)]] error: attempt to select more than one element vectorIndex
# ------------------------------------------------
# Empty brackets [] will return original vector
# ------------------------------------------------
x[]
```

```
##   one   two three  four
##   2.1   4.2   5.3   1.4
```

```
# x[[]] error: subscript out of bound
# --------------------------------
# 0 will return zero length vector
# --------------------------------
x[0]
```

```
## named numeric(0)
```

```
# x[[0]] error: attempt to select more than one element vectorIndex


#------------ Create an empty vector of given length ------------------
# vector("type", length = <<length>>)
# <<type>>(<<length>>) like numeric(3)
# ----------------------------------------------------------------------

identical(numeric(0) , vector("double" , length = 0))
```

```
## [1] TRUE
```

```
identical(logical(2) , vector("logical", length = 2))
```

```
## [1] TRUE
```

```
identical(integer(0) , vector("integer", length = 0))
```

```
## [1] TRUE
```

```
identical(complex(0) , vector("complex", length = 0))
```

```
## [1] TRUE
```

```
identical(character(4) , vector("character", length = 4))
```

```
## [1] TRUE
```

```
# ------------------------------------------------------------------
# [[]] with zero length, NULL and out of bound index or name return error
# ------------------------------------------------------------------
x[NULL]
```

```
## named numeric(0)
```

```
# x[[NULL]]  error: attempt to select less than one element in get1index

x[logical()]
```

```
## named numeric(0)
```

```r
# x[[logical()]] error: attempt to select less than one element in get1index

x[1000]
```

```
## <NA>
##   NA
```

```r
# x[[1000]] error: subscript out of bound

x["out_of_bound"]
```

```
## <NA>
##   NA
```

```r
# x[["out of bound"]] error: subscript out of bound

#----------------- NA as index ---------------------
# NA has logical type and logical vector is recycled
#-------------------------------------------------
x[NA]
```

```
## <NA> <NA> <NA> <NA>
##   NA   NA   NA   NA
```

```r
# x[[NA]] error: subscript out of bound
# ------------------------------------------
# named vectors can also be subset with names
# ------------------------------------------
x[c("one","three","one","four")]
```

```
##   one three   one  four
##   2.1   5.3   2.1   1.4
```

```r
# x[[c("one","three","one","four")]] # error: attempt to select more than one element vectorIndex
# ------------------------------------------------------------------------
# factor subsetting is based on underlying integer vector not the levels
# drop=T in operator [] controls if levels are dropped
# ------------------------------------------------------------------------
y <- ordered(c("a","b","b","a", "c"), levels = c("c","b","a"))
y[c(1,2,3)]
```

```
## [1] a b b
## Levels: c < b < a
```

```r
# y[[c(1,2,3)]] # error: attempt to select more than one element vectorIndex


y[1]
```

```
## [1] a
## Levels: c < b < a
```

```r
y[[1]] # recomended
```

```
## [1] a
## Levels: c < b < a
```

```r
y[1, drop=T]
```

```
## [1] a
## Levels: a
```

```
y[[1, drop=T]] # recomended
```

```
## [1] a
## Levels: c < b < a
```

List Subsettings

- Key Pints:
  - *typeof()* applied on a list returns *list*
  - *[ ]* operator always returns a list whereas *[[ ]]* operator returns a single object
  - *$*col is a short form for *[["col"]]* returns a single object bound to "col"
  - *$*col is translated to *[["col"]]* by R
  - *[[ ]]* better be used with single positive integer or single string
  - Subsetting a non_empty list with *[[ ]]* returns:
    - list[[NULL]] ==> Error
    - list[[integer(0)]] ==> Error
    - list[[out_of_bound_integer]] ==> NULL
    - list[[out_of_bound_character]] ==> NULL
  - NULL is an empty list
  - Subsetting NULL always returns NULL
    - NULL[[NULL]] ==> NULL
    - NULL[[integer(0)]] ==> NULL
    - NULL[[out_of_bound integer]] ==> NULL
    - NULL[[out_of_bound character]] ==> NULL
    - NULL[NULL] ==> NULL
    - NULL[integer(0)] ==> NULL
    - NULL[out_of_bound_integer] ==> NULL
    - NULL[out_of_bound_character] ==> NULL
  - *[[c(1,2)]]* is equivalent to *[[1]][[2]]* (use *purrr::pluck(x, 1, 2)* instead)
  - *$* operator does partial matching but *[[ ]]* operator performs full matching
  - avoid silent partial matching by: options(warnPartialMatchDollar = T)
  - To create an empty list: *list()*
  - To create a list of length n full of NULL: *vector("list", length=n)*
  - *NULL == list() == list()[0] == list()[1]*
  - To convert a list to a vector use *unlist* (note *vector(list)* does not work!!)
  - Note that *unlist* strip off attributes of the object in the list
  - *list[NA]* returns *NA*

```
# ------------------------------------------------------------
# ----------- How R shows a named list ---------------------
# ------------------------------------------------------------
(x <- list(a = 1, b=list(2,3,NULL), c=c(4,5,6) ))
```

```
## $a
## [1] 1
##
## $b
## $b[[1]]
## [1] 2
##
## $b[[2]]
## [1] 3
```

```
##
## $b[[3]]
## NULL
##
##
## $c
## [1] 4 5 6
```

```
# ------------------------------------------------------------
# ----------- How R shows an unnamed list ------------------
# ------------------------------------------------------------
(x <- list(1, list(2,3,NULL), c(4,5,6) ))
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [[2]][[1]]
## [1] 2
##
## [[2]][[2]]
## [1] 3
##
## [[2]][[3]]
## NULL
##
##
## [[3]]
## [1] 4 5 6
```

```
# ------------------------------------------------------------------------------
# ---- [] always returns a list, [[]] and $ returns elements in the list ----->
# ------------------------------------------------------------------------------
(x <- list(a=list(1,2,3,4,5,6,7,8), b = c(9,10,11), d = 13))
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
## $a[[3]]
## [1] 3
##
## $a[[4]]
## [1] 4
##
## $a[[5]]
## [1] 5
##
## $a[[6]]
## [1] 6
##
## $a[[7]]
## [1] 7
```

```
##
## $a[[8]]
## [1] 8
##
##
## $b
## [1]  9 10 11
##
## $d
## [1] 13
```

```
# --------------------------------
# -------- by position ----------
# --------------------------------
x[3]
```

```
## $d
## [1] 13
```

```
x[[3]]
```

```
## [1] 13
```

```
x$d
```

```
## [1] 13
```

```
feature.name <- "d"
x$feature.name # this returns NULL because it translates to x[["feature.name"]]
```

```
## NULL
```

```
x[[feature.name]]
```

```
## [1] 13
```

```
x[c(1,2)]
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
## $a[[3]]
## [1] 3
##
## $a[[4]]
## [1] 4
##
## $a[[5]]
## [1] 5
##
## $a[[6]]
## [1] 6
##
## $a[[7]]
## [1] 7
```

```
##
## $a[[8]]
## [1] 8
##
##
## $b
## [1]  9 10 11
```

```
x[[c(1,8)]] # == x[[1]][[8]]
```

```
## [1] 8
```

```
purrr::pluck(x,1,8)
```

```
## [1] 8
```

```
x[[1]][[8]]
```

```
## [1] 8
```

```
x$a[[8]]
```

```
## [1] 8
```

```
# ------------------------------------------------------------
# $ does the partial matching but [["col"]] does full maching
# to get a warning when R does partial matching always set
#          options(warnPartialMatchDollar = T)
# ------------------------------------------------------------

options(warnPartialMatchDollar = T)
(x <- list(acd=list(1,2), adc=list(1,2), d = 13))
```

```
## $acd
## $acd[[1]]
## [1] 1
##
## $acd[[2]]
## [1] 2
##
##
## $adc
## $adc[[1]]
## [1] 1
##
## $adc[[2]]
## [1] 2
##
##
## $d
## [1] 13
```

```
x$ac
```

```
## Warning in x$ac: partial match of 'ac' to 'acd'
```

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

```
## [[2]]
## [1] 2
```

```
(x <- list(a=list(1,2), b = c(9,10,11), d = 13))
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
##
## $b
## [1]  9 10 11
##
## $d
## [1] 13
```

```
# ------------------------------------
# duplicate index means duplicate values
# ------------------------------------
x[c(2,1,1)]
```

```
## $b
## [1]  9 10 11
##
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
##
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
```

```
# ------------------------------------------------------------------------
# order() not implemented for list
# ------------------------------------------------------------------------


(x <- list(a=list(1,2), b = c(9,10,11), d = 13))
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
```

```
##
## $b
## [1]  9 10 11
##
## $d
## [1] 13
```

```
# ------------------------------------------------------
# real numbers used as indices are truncated to integers
# ------------------------------------------------------

x[c(3.1, 1.2)]
```

```
## $d
## [1] 13
##
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
```

```
(x <- list(a=list(1,2), b = c(9,10,11), d = 13))
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
##
## $b
## [1]  9 10 11
##
## $d
## [1] 13
```

```
# ------------------------------------------------------------
# use negative elements to exclude values at specified position
# ------------------------------------------------------------
x[-2]
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
##
## $d
## [1] 13
```

```
x[c(-3:-1)]
```

```
## named list()
```

```
(x <- list(a=list(1,2), b = c(9,10,11), d = 13))
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
##
## $b
## [1]  9 10 11
##
## $d
## [1] 13
```

```
# ------------------------------------------------------------------
# Logical vectors as index makes R chooses elements with TRUE as index
# ------------------------------------------------------------------

x[c(T,F,F)]
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
```

```
# x[x>3] # # Error: 'list' object cannot be coerced to type 'double'


(x <- list(a=list(1,2), b = c(9,10,11), d = 13))
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
##
## $b
## [1]  9 10 11
##
## $d
## [1] 13
```

```
# ---------------------------------
# # Only logical index is recycled
# ---------------------------------
x[T]
```

```
## $a
## $a[[1]]
```

```
## [1] 1
##
## $a[[2]]
## [1] 2
##
##
## $b
## [1]  9 10 11
##
## $d
## [1] 13
```

```
(x <- list(a=list(1,2), b = c(9,10,11), d = 13))
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
##
## $b
## [1]  9 10 11
##
## $d
## [1] 13
```

```
# --------------------------------
# NA in index causes NA in output
# --------------------------------
x[c(2,3,NA,4)]
```

```
## $b
## [1]  9 10 11
##
## $d
## [1] 13
##
## $<NA>
## NULL
##
## $<NA>
## NULL
```

```
(x <- list(a=list(1,2), b = c(9,10,11), d = 13))
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
##
## $b
```

```
## [1]  9 10 11
##
## $d
## [1] 13
```

```
# -------------------------------------------
# Empty brackets [] will return original list
# -------------------------------------------
x[]
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
##
## $b
## [1]  9 10 11
##
## $d
## [1] 13
```

```
(x <- list(a=list(1,2), b = c(9,10,11), d = 13))
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
##
## $b
## [1]  9 10 11
##
## $d
## [1] 13
```

```
# --------------------------------
# 0 will return zero length list
# --------------------------------
x[0]
```

```
## named list()
```

```
(x <- list(a=list(1,2), b = c(9,10,11), d = 13))
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
##
```

```
## $b
## [1]  9 10 11
##
## $d
## [1] 13
```

```
# ------------------------------------------------------------------------
#  Subestting with NULL OOB (int), OOB (char) and NA
# ------------------------------------------------------------------------

x[numeric()]
```

```
## named list()
```

```
#  x[[numeric()]] Error : attempt to select less than one element in get1index

x[NULL]
```

```
## named list()
```

```
#  x[[NULL]] Error : attempt to select less than one element in get1index

x[1000]
```

```
## $<NA>
## NULL
```

```
#x[[1000]] Error : subscript out of bounds

x["out_of_bound"]
```

```
## $<NA>
## NULL
```

```
x[["out_of_bound"]]
```

```
## NULL
```

```
#----------------- NA as index ---------------------
# NA has logical type and logical vector is recycled
#-------------------------------------------------
x[NA]
```

```
## $<NA>
## NULL
##
## $<NA>
## NULL
##
## $<NA>
## NULL
```

```
x[[NA]]
```

```
## NULL
```

```
# ------------------------------------------------------------------------
#  NULL is an empty list. Subsetting NULL with [[]] and [] always returns NULL
# ------------------------------------------------------------------------
NULL[[1]]
```

```
## NULL
```

```
NULL[[]]
```

```
## NULL
```

```
NULL[[numeric(0)]]
```

```
## NULL
```

```
NULL[[NULL]]
```

```
## NULL
```

```
NULL[["out_of_bound"]]
```

```
## NULL
```

```
NULL[[1000]]
```

```
## NULL
```

```
NULL[1]
```

```
## NULL
```

```
NULL[]
```

```
## NULL
```

```
NULL[numeric(0)]
```

```
## NULL
```

```
NULL[NULL]
```

```
## NULL
```

```
NULL["out_of_bound"]
```

```
## NULL
```

```
NULL[1000]
```

```
## NULL
```

```
(x <- list(a=list(1,2), b = c(9,10,11), d = 13))
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
##
## $b
## [1]  9 10 11
##
## $d
## [1] 13
```

```
# ------------------------------------------
# named vectors can also be subset with names
```

```r
# ----------------------------------------
x[c("a","b","d")]
```

```
## $a
## $a[[1]]
## [1] 1
##
## $a[[2]]
## [1] 2
##
##
## $b
## [1]  9 10 11
##
## $d
## [1] 13
```

```r
# ------------------------------------
# We can construct a factor from a list
# ------------------------------------
(y <- ordered(list("a","b","b","a", "c"), levels = c("c","b","a")))
```

```
## [1] a b b a c
## Levels: c < b < a
```

```r
y[c(1,2,3)]
```

```
## [1] a b b
## Levels: c < b < a
```

```r
# ------------------------------------------------------------------
# purrr::pluck always returns NULL when OOB or empty index object
# ------------------------------------------------------------------
purrr::pluck(x, "a", 2)
```

```
## [1] 2
```

```r
purrr::pluck(x, 2, 3)
```

```
## [1] 11
```

```r
purrr::pluck(x, 3)
```

```
## [1] 13
```

```r
purrr::pluck(x, 3, 1)
```

```
## [1] 13
```

```r
purrr::pluck(x, "Out_of_Bound", 2)
```

```
## NULL
```

```r
purrr::pluck(x, "b", 1000)
```

```
## NULL
```

```r
(l <- list(x=list(1,2,3,4), y = list(5,6,7,8), z = list(9,10,11,12)))
```

```
## $x
## $x[[1]]
```

```
## [1] 1
##
## $x[[2]]
## [1] 2
##
## $x[[3]]
## [1] 3
##
## $x[[4]]
## [1] 4
##
##
## $y
## $y[[1]]
## [1] 5
##
## $y[[2]]
## [1] 6
##
## $y[[3]]
## [1] 7
##
## $y[[4]]
## [1] 8
##
##
## $z
## $z[[1]]
## [1] 9
##
## $z[[2]]
## [1] 10
##
## $z[[3]]
## [1] 11
##
## $z[[4]]
## [1] 12
```

```r
# ---------------------------------------
# get first element of each inner list
# ---------------------------------------

lapply(l, function(inner.list) inner.list[[1]])
```

```
## $x
## [1] 1
##
## $y
## [1] 5
##
## $z
## [1] 9
```

```r
suppressWarnings(suppressMessages(library(tidyverse)))
library(tidyverse)
map(l, 1)
```

```
## $x
## [1] 1
##
## $y
## [1] 5
##
## $z
## [1] 9
```

```r
# ------------------------------------------------------------------------
# ------ note unlist strips off attributes of the object inside the list ----
# ------------------------------------------------------------------------
(l <- list(as.Date("1980-01-02")))
```

```
## [[1]]
## [1] "1980-01-02"
```

```r
unlist(l)
```

```
## [1] 3653
```

```r
(l1 <- rerun(2, sample(4)))
```

```
## [[1]]
## [1] 1 2 4 3
##
## [[2]]
## [1] 1 2 4 3
```

```r
typeof(l1)
```

```
## [1] "list"
```

```r
flatten_int(l1)
```

```
## [1] 1 2 4 3 1 2 4 3
```

Matrix Subsettings

- Key Pints:
  - *matrix* is just an atomic vector with three attributes:
    - dimensions
    - row names
    - column names
  - Subset a *matrix* by supplying a vector of indices for each dimension:
    - All rules for subsetting a vector is applied to subsetting each dimension
  - vector of values is used to fill a *matrix* up column by column or row by row
  - Dimension Dropping:
    - *[ ]* simplifies result to loweset possible dimensionality unless we set *drop=F*
    - *matrix[row #i,]* retuns row #i as a vector
    - *matrix[,col# j]* retuns col #j as a vector
    - To preserve dimensionality set *drop=F* as last argument of operator *[[ ]]*
  - Subsetting a *matrix* with a single vector

        * *matrix* is a vector so like vectors we can use vector of indices to subset it
        * Index of the elements in the *matrix* always goes column by column
    - Subsetting *matrix* with logical *matrix* which has same dimension as the original one
        * R selects those elements that correspond to TRUE in logical matrix

```
(a <- matrix(1:9, nrow = 3, byrow = F))
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
colnames(a) <- c("A", "B", "C")

str(a)
```

```
##  int [1:3, 1:3] 1 2 3 4 5 6 7 8 9
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:3] "A" "B" "C"
```

```
# ------------------------------------------------------------------
# subsetting by supplying a vector of indices for each dimension
# ------------------------------------------------------------------

a[0, -2]  # remember 0 as index returns an empty vector or list
```

```
##      A C
```

```
a[1:2,]
```

```
##      A B C
## [1,] 1 4 7
## [2,] 2 5 8
```

```
a[c(T,T,F), c("A", "C")]
```

```
##      A C
## [1,] 1 7
## [2,] 2 8
```

```
# ----------------------- Dimension Dropping -----------------------

a[2, ]
```

```
## A B C
## 2 5 8
```

```
a[2, , drop=F]
```

```
##      A B C
## [1,] 2 5 8
```

```
a[, 3]
```

```
## [1] 7 8 9
```

```
a[, 3, drop=F]
```

```
##      C
## [1,] 7
```

```
## [2,] 8
## [3,] 9
```

```
a[1,1]
```

```
## A
## 1
```

```
a[1,1, drop=F]
```

```
##      A
## [1,] 1
```

```
# ---------------- Subsetting a matrix with a single vector -------------------

(a <- matrix(1:4, nrow = 2, byrow = T))
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

```
colnames(a) <- c("A", "B")


a[1] # => 1
```

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

```
a[2] # => 3
```

```
## [1] 3
```

```
a[3] # => 2
```

```
## [1] 2
```

```
a[4] # => 4
```

```
## [1] 4
```

```
a[c(2,4)]
```

```
## [1] 3 4
```

```
(a <- matrix(1:4, nrow = 2, byrow = F))
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
colnames(a) <- c("A", "B")

a[1] # => 1
```

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

```
a[2] # => 2
```

```
## [1] 2
```

```
a[3] # => 3
```

```
## [1] 3
```

```r
a[4] # => 4
```

```
## [1] 4
```

```r
a[c(2,4)]
```

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

```r
# ------------ Subsetting with logical matrix ----------------------------
(a <- outer(1:5, 1:5, FUN="*"))
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    2    4    6    8   10
## [3,]    3    6    9   12   15
## [4,]    4    8   12   16   20
## [5,]    5   10   15   20   25
```

```r
selection.matrix <- upper.tri(a)

a[selection.matrix]
```

```
##  [1]  2  3  6  4  8 12  5 10 15 20
```

```r
# Another example of logical matrix
a <- outer(1:5, 1:5, FUN="*")
diagonal.selection <- matrix(rep(F, 25), c(5,5))
(for (i in 1:5)
  for (j in 1:5)
    diagonal.selection[i,j] <- (i==j)
)
```

```
## NULL
```

```r
diagonal.selection
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]
## [1,]  TRUE FALSE FALSE FALSE FALSE
## [2,] FALSE  TRUE FALSE FALSE FALSE
## [3,] FALSE FALSE  TRUE FALSE FALSE
## [4,] FALSE FALSE FALSE  TRUE FALSE
## [5,] FALSE FALSE FALSE FALSE  TRUE
```

```r
a[diagonal.selection]
```

```
## [1]  1  4  9 16 25
```

```r
# Note:
# a$A Error in a$A : $ operator is invalid for atomic vectors
```

Dataframe Subsettings as List of Lists and as Matrix

- Key Pints:
  - Dataframe treated as a list of lists:
    - When subsetting with a single index or vector of indices it behaves like a list of columns (i.e list of lists)
    - All rules for list subsetting using *[ ]* and *[[ ]]* are applied
    - Single index *[ ]* returns a dataframe (i.e a list, remember list subsetting rules)

    - Single index *[[ ]]* returns vector (i.e an object , remember list subsetting rules)
  - Dataframe treated as a *matrix*
    - Two sets of indices are used to subset a dataframe which returns a dataframe
    - Filter condition is always filter rows ,it is only on first index
    - dataframe with single column returns only that column unless we use drop=F (remember dimension dropping)

```
df <- data.frame(x = 1:3, y=3:1, z = letters[1:3])

# ----------------- Dataframe: a list of lists perspective ---------------------

df[2]  # => dataframe containing second column
```

```
##   y
## 1 3
## 2 2
## 3 1
```

```
df[[1]]  # =>  second column as a vector
```

```
## [1] 1 2 3
```

```
df [c(1,3)] # dataframe containg first and third column
```

```
##   x z
## 1 1 a
## 2 2 b
## 3 3 c
```

```
df [[c(1,3)]] # third element in first column (see the list subsetting)
```

```
## [1] 3
```

```
df[c("x", "z")] # dataframe containg first and third column
```

```
##   x z
## 1 1 a
## 2 2 b
## 3 3 c
```

```
df [c(2,2,1)] # dataframe contatining two times column #2 (named as Y and Y.1) and column #1
```

```
##   y y.1 x
## 1 3   3 1
## 2 2   2 2
## 3 1   1 3
```

```
# ----------------- Dataframe: a matrix perspective ---------------------

df[df$x == 2, ] # => dataframe containing a row(s) with value 2 in its x column
```

```
##   x y z
## 2 2 2 b
```

```
df[c(1,3), ] # dataframe containing row# 1 and row # 3
```

```
##   x y z
## 1 1 3 a
## 3 3 1 c
```

```r
df[c(2,2,1), ] # dataframe containing two times row #2 () and row # 1
```

```
##     x y z
## 2   2 2 b
## 2.1 2 2 b
## 1   1 3 a
```

```r
df [, c(2,2,1)] # dataframe contatining two times column #2 (named as Y and Y.1) and column #1
```

```
##   y y.1 x
## 1 3   3 1
## 2 2   2 2
## 3 1   1 3
```

```r
df[, 1] # remember from matrices that matrix[ ,col# j] retuns col #j as a vector
```

```
## [1] 1 2 3
```

```r
df[, 1, drop=F] # dataframe with single column #1
```

```
##   x
## 1 1
## 2 2
## 3 3
```

```r
df[, "y"]
```

```
## [1] 3 2 1
```

```r
df[, "y", drop=F] # dataframe with single column "y"
```

```
##   y
## 1 3
## 2 2
## 3 1
```

```r
suppressWarnings(suppressMessages(library(tidyverse)))
library(tidyverse)

# ---------- Cool tidyverse stuff --------------#
(df <- tibble(x = c("  "," 1  3", "12", "14  ",NA,"  "," 1  3", "12", "14  ", NA),
              y = c(1,2,NA,4,5,1,2,NA,4,5)))
```

```
## # A tibble: 10 x 2
##    x           y
##    <chr>   <dbl>
##  1 "  "        1
##  2 " 1  3"     2
##  3 "12"       NA
##  4 "14  "      4
##  5  <NA>       5
##  6 "  "        1
##  7 " 1  3"     2
##  8 "12"       NA
##  9 "14  "      4
## 10  <NA>       5
```

```r
# First remove NA and empty spaces only from character columns
df %>%
```

```
  filter_if(is.character, any_vars(!is.na(.) & trimws(.) != ""))
```

```
## # A tibble: 6 x 2
##   x           y
##   <chr>   <dbl>
## 1 " 1  3"     2
## 2 "12"       NA
## 3 "14  "      4
## 4 " 1  3"     2
## 5 "12"       NA
## 6 "14  "      4
```

```
# Next remove leading and trailing spaces from all elements in character columns
trim.f <- function(col, na.rm = F) {
  isNA <- !reduce(col, ~ (is.na(.x) & is.na(.y)))
  if (na.rm && isNA)
    unlist(map(col, ~ (if (is.na(.x)) "" else .x) ),use.names = F) %>%
      trimws(which = c("both")) # leading and trailing spaces
  else trimws(col, which = c("both")) # leading and trailing spaces
}

(df %>%
  mutate_if(is.character, trim.f, na.rm = T))
```

```
## # A tibble: 10 x 2
##    x          y
##    <chr>  <dbl>
##  1 ""         1
##  2 "1  3"     2
##  3 "12"      NA
##  4 "14"       4
##  5 ""         5
##  6 ""         1
##  7 "1  3"     2
##  8 "12"      NA
##  9 "14"       4
## 10 ""         5
```

```
# pull(col) has the same effect as $col on dataframe
identical(df %>% pull(x), df$x)
```

```
## [1] TRUE
```

```
# Finally convert character columns to factor

df %>%
  filter_if(is.character, any_vars(!is.na(.) & trimws(.) != "")) %>%
  mutate_if(is.character, trim.f, na.rm = T) %>%
  mutate_if(is.character, ~ ordered(.x, levels = unlist(.x %>% table %>% dimnames)))
```

```
## # A tibble: 6 x 2
##   x         y
##   <ord> <dbl>
## 1 1  3      2
## 2 12       NA
## 3 14        4
```

```
## 4 1  3      2
## 5 12       NA
## 6 14        4
```

```
# distinct values of certain columns
df %>% distinct(x,y)
```

```
## # A tibble: 5 x 2
##   x           y
##   <chr>   <dbl>
## 1 "  "        1
## 2 " 1  3"     2
## 3 "12"       NA
## 4 "14  "      4
## 5  <NA>       5
```

```
# select only character columns
# df %>%
#   select_if(~is.character(.))
#   filter (!is.na(x) & trimws(x) != "")

# msleep %>%
#   select(name:order, sleep_total:sleep_rem) %>%
#   filter_if(is.character, any_vars(is.na(.)))

# -------------------------
```

```
suppressWarnings(suppressMessages(library(tidyverse)))
library(tidyverse)
gapminder_orig <-
  read.csv(
    "https://raw.githubusercontent.com/swcarpentry/r-novice-gapminder/gh-pages/_episodes_rmd/data/gapmi

gapminder <-gapminder_orig
# get dataframe schema
gapminder %>%
  map_chr(class)
```

```
##   country      year       pop continent   lifeExp gdpPercap
##  "factor" "integer" "numeric"  "factor" "numeric" "numeric"
```

```
# get number of distince elements in each column
gapminder %>%
  map_dbl(n_distinct)
```

```
##   country      year       pop continent   lifeExp gdpPercap
##       142        12      1704         5      1626      1704
```

```
# get number of NA in each column
gapminder %>%
  map_dbl(~ sum(is.na(.)))
```

```
##   country      year       pop continent   lifeExp gdpPercap
##         0         0         0         0         0         0
```

```
# get a summary of each column as a dataframe
gapminder %>%
  map_df(~ tibble(type=class(.),
```

```
                  no.of.elements = n_distinct(.),
                  nas = sum(is.na(.))),
         .id="variable")
```

```
## # A tibble: 6 x 4
##   variable  type    no.of.elements   nas
##   <chr>     <chr>            <int> <int>
## 1 country   factor             142     0
## 2 year      integer             12     0
## 3 pop       numeric           1704     0
## 4 continent factor               5     0
## 5 lifeExp   numeric           1626     0
## 6 gdpPercap numeric           1704     0
```

```
gapminder %>% sample_n(5) %>% pluck(1) # get the first column of dataframe
```

```
## [1] Cameroon              Nicaragua             Saudi Arabia
## [4] Sao Tome and Principe Somalia
## 142 Levels: Afghanistan Albania Algeria Angola Argentina Australia ... Zimbabwe
```

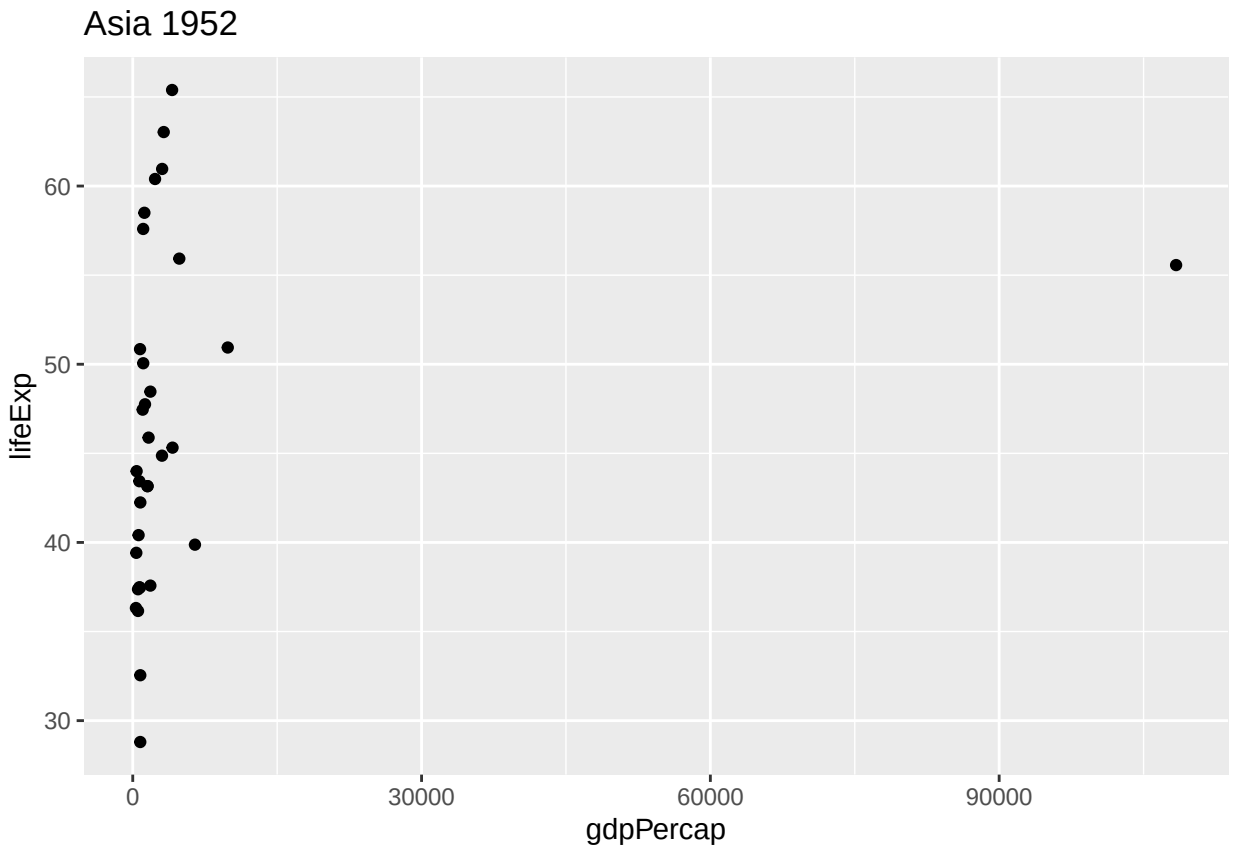
```
# map2 help us with zip
map2(1:5, 1:5, ~ c(.x, .y))
```

```
## [[1]]
## [1] 1 1
##
## [[2]]
## [1] 2 2
##
## [[3]]
## [1] 3 3
##
## [[4]]
## [1] 4 4
##
## [[5]]
## [1] 5 5
```

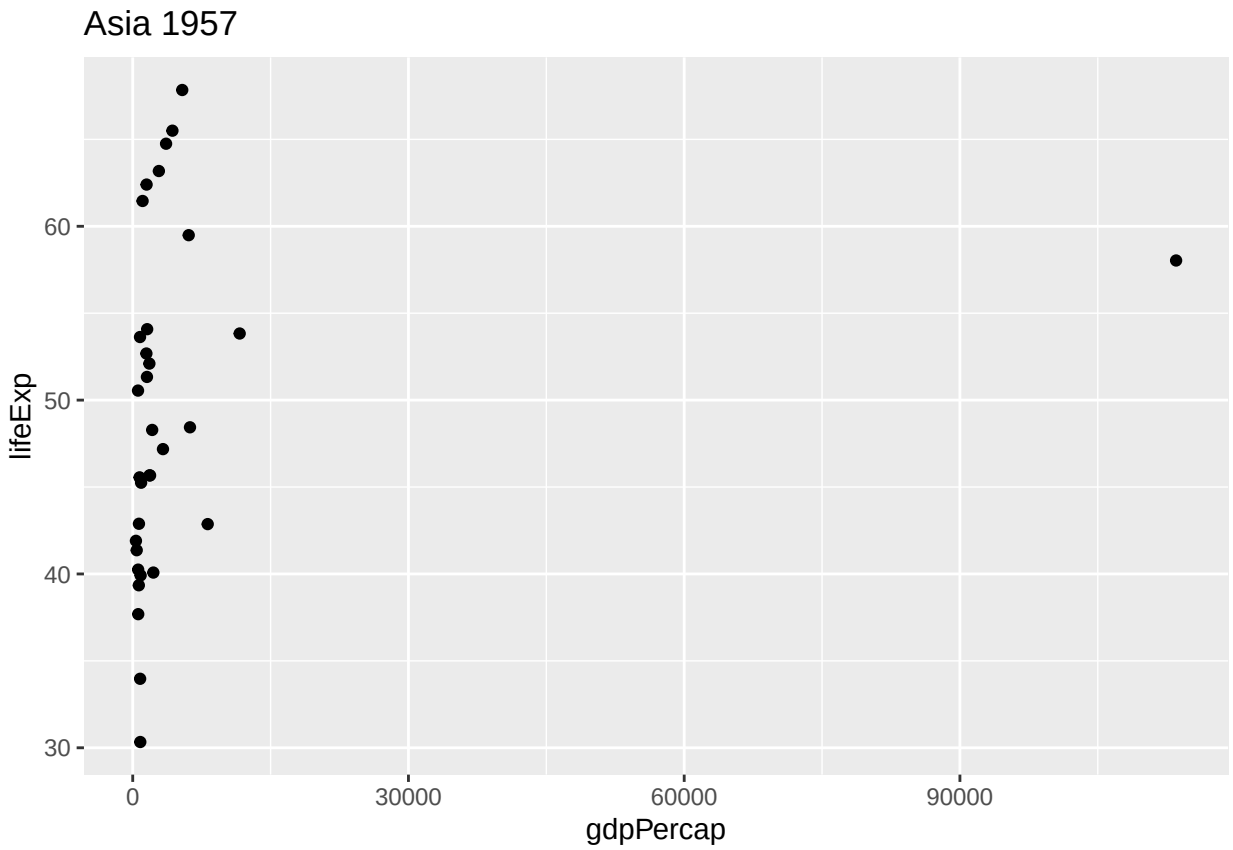
```
plot_it <- function(df) {
  distincts.pairs <- df %>% distinct(continent, year)
  map2(distincts.pairs %>% pull(continent) %>% as.character,
       distincts.pairs %>% pull(year) ,
       ~ df %>%
           filter(continent == .x, year == .y) %>%
           ggplot() +
           geom_point(aes(x = gdpPercap, y = lifeExp)) +
           ggtitle(glue::glue(.x, " ", .y)))
}

gapminder %>% plot_it
```

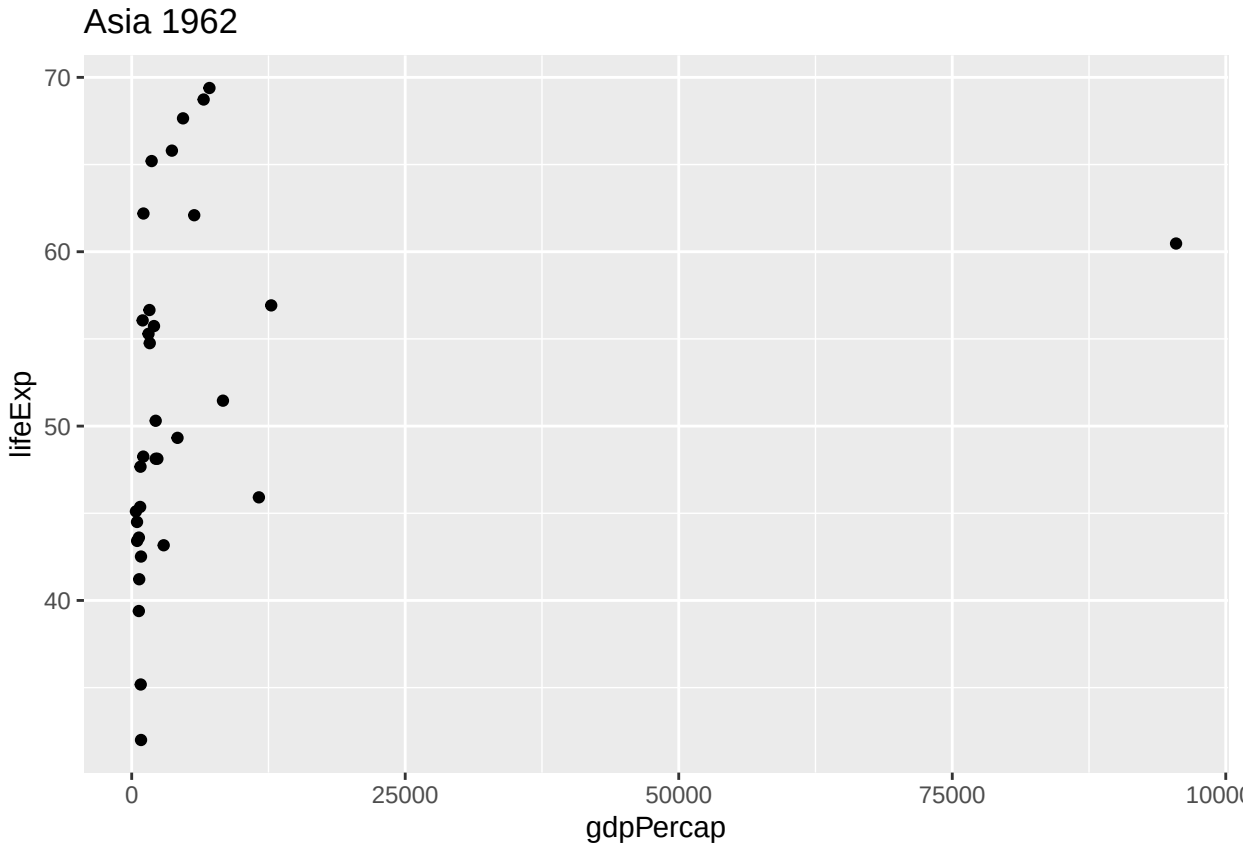
```
## [[1]]
```



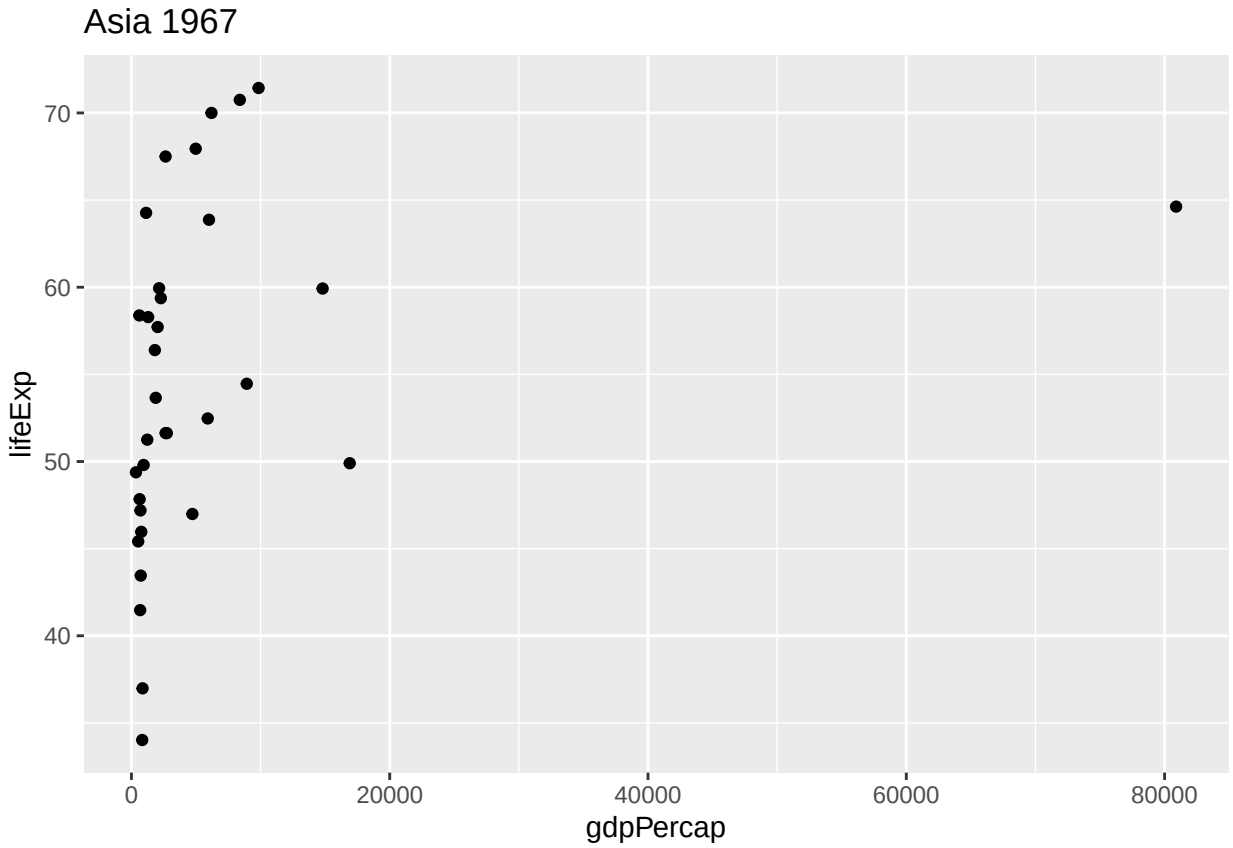

```
##
## [[2]]
```



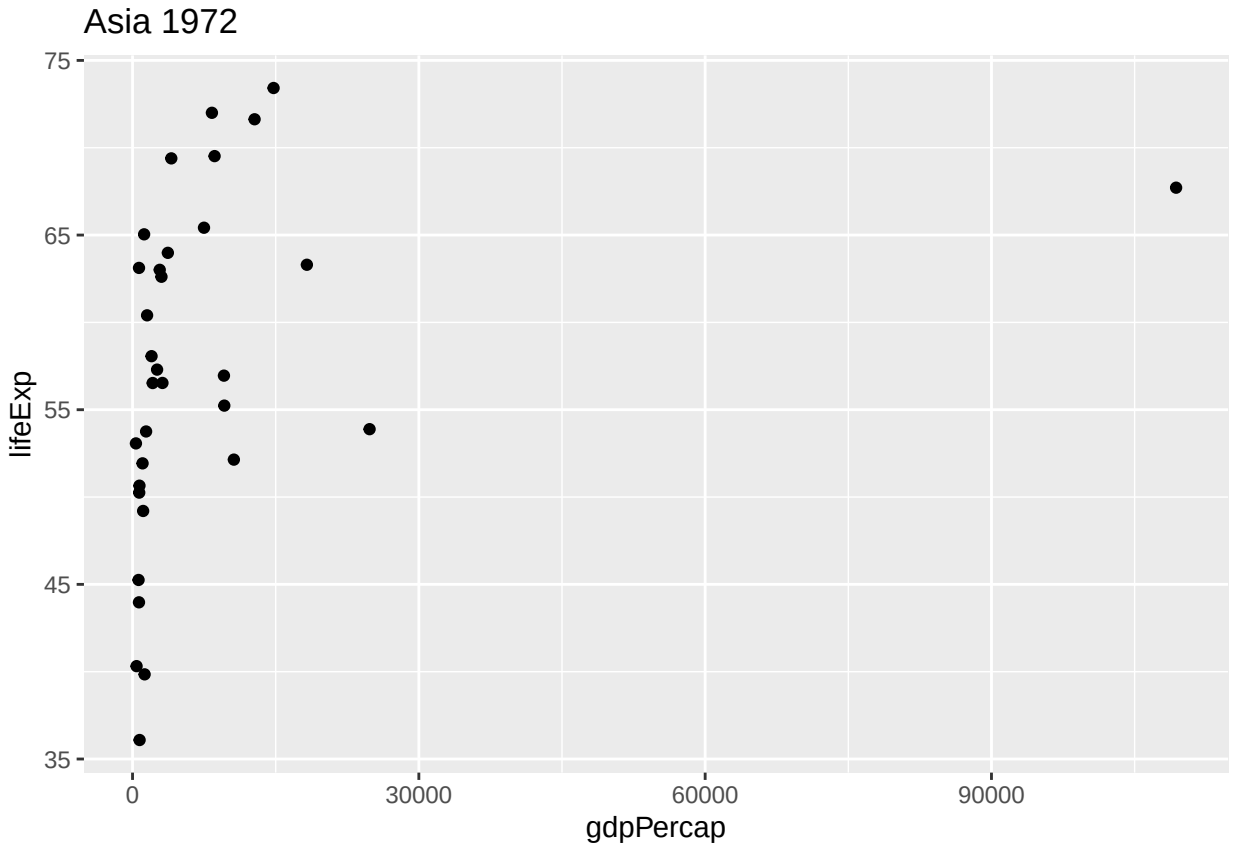

```
##
## [[3]]
```



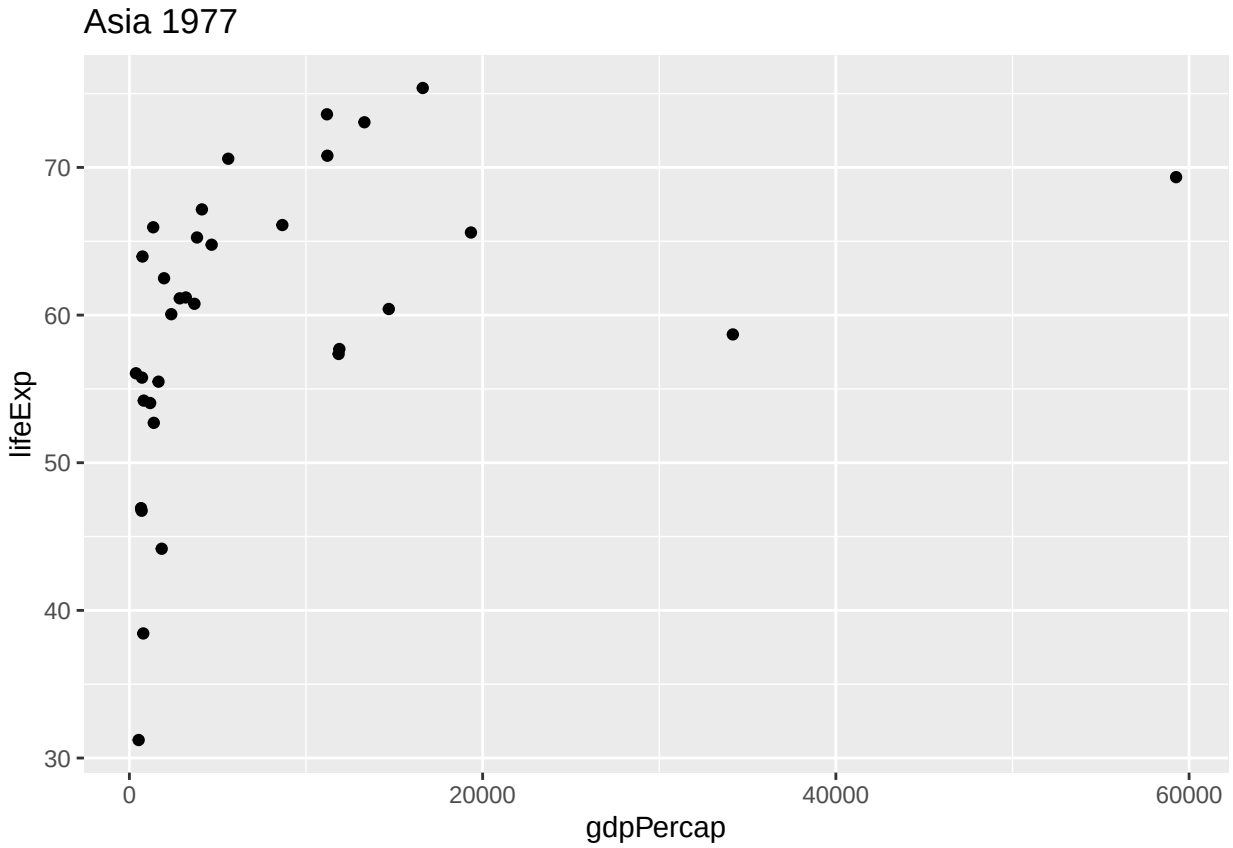

```
##
## [[4]]
```


Asia 1967
lifeExp
70
60
50
40
0
20000
40000
60000
80000
gdpPercap

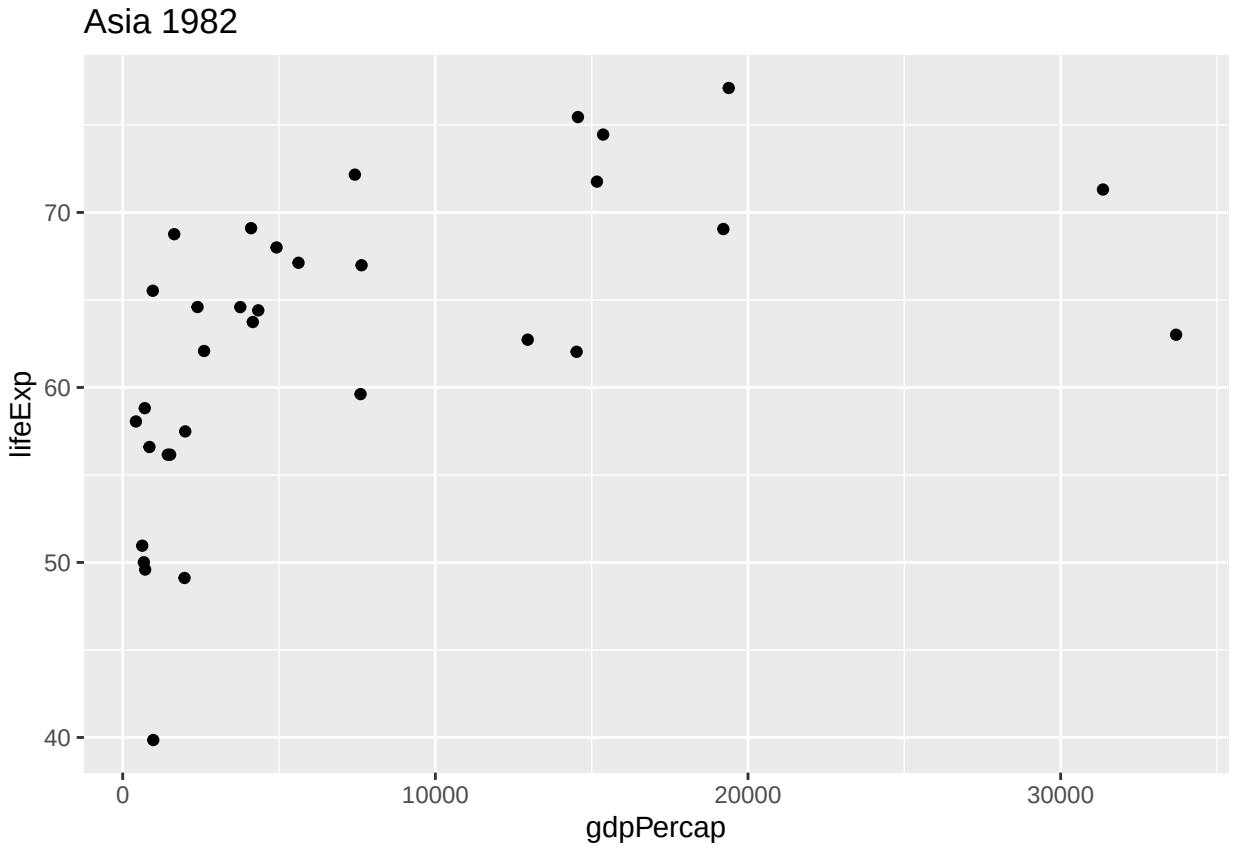
```
##
## [[5]]
```


Asia 1972
lifeExp
75
65
55
45
35
0
30000
60000
90000
gdpPercap

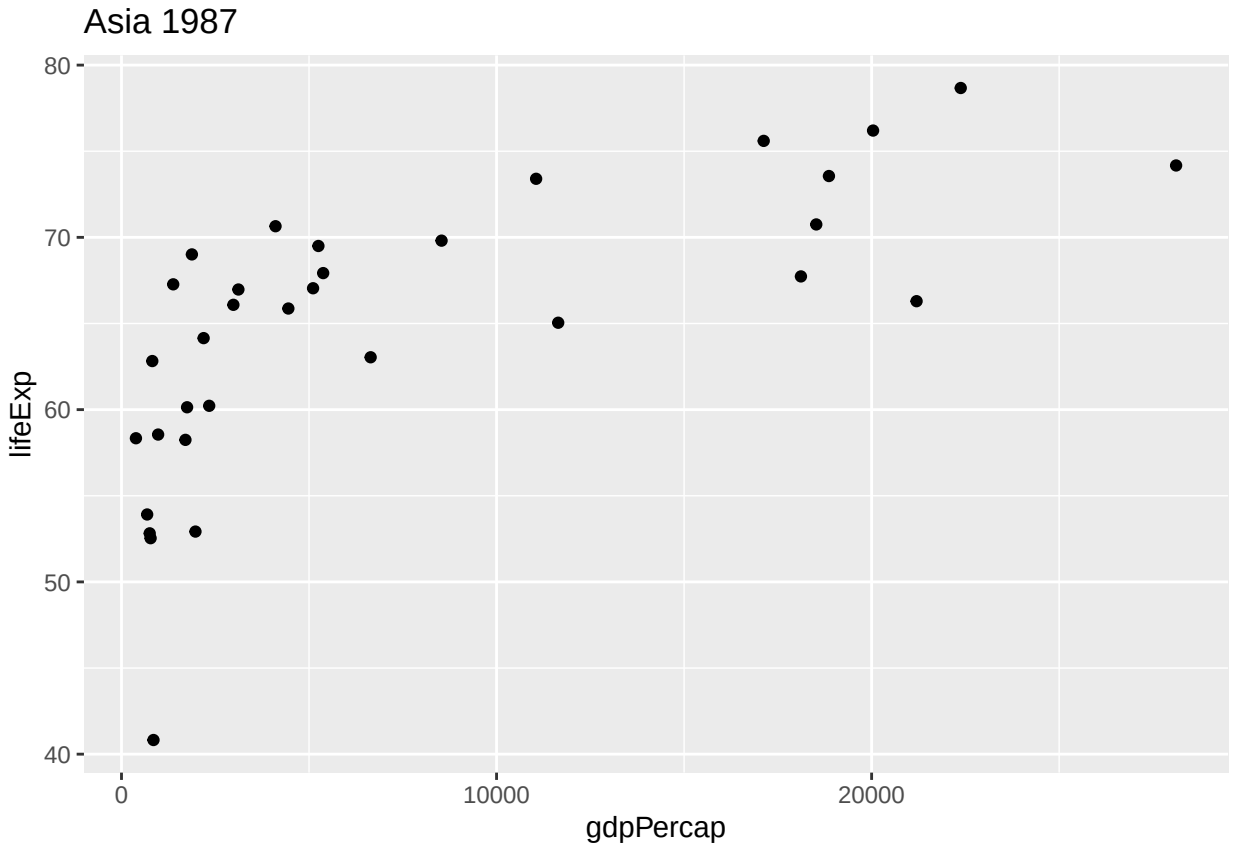
```
##
## [[6]]
```


Asia 1977
lifeExp
70
60
50
40
30
0
20000
40000
60000
gdpPercap

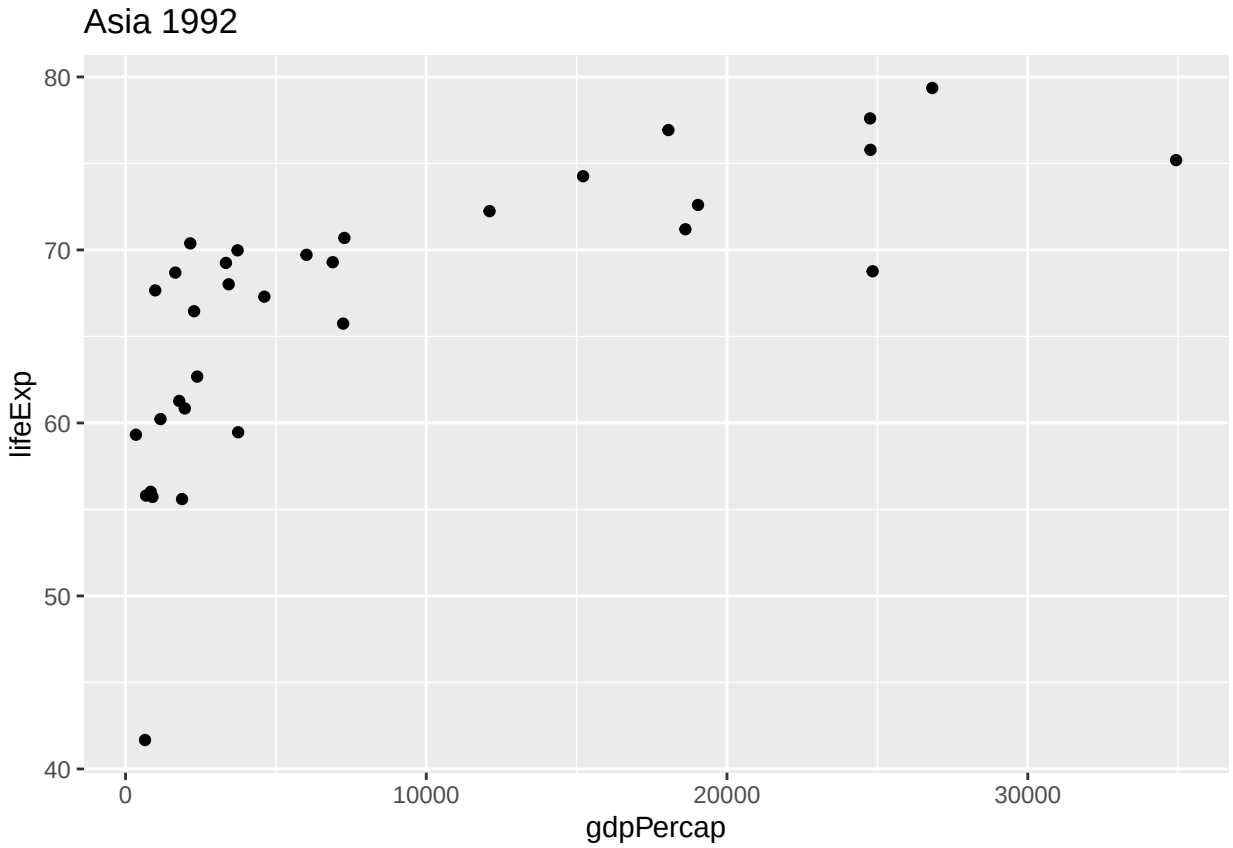
```
##
## [[7]]
```



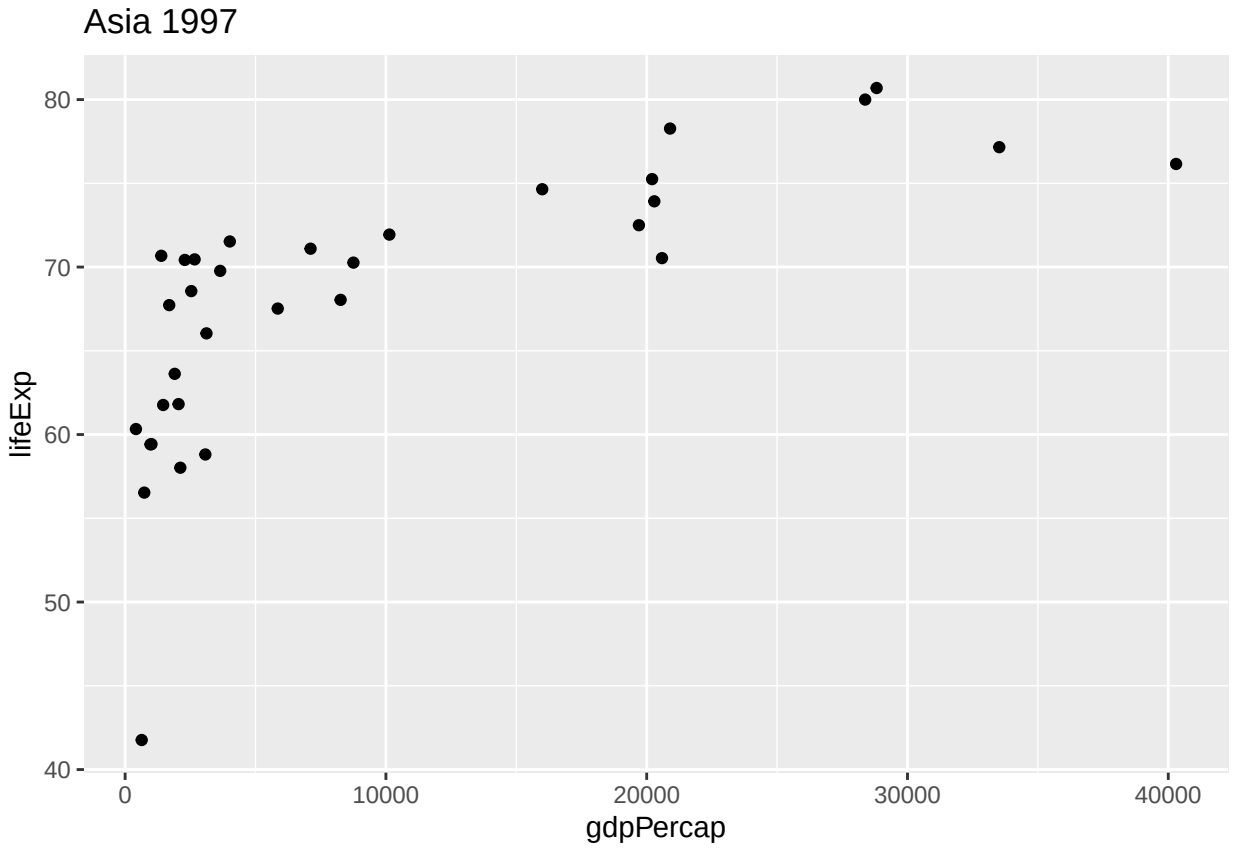

```
##
## [[8]]
```



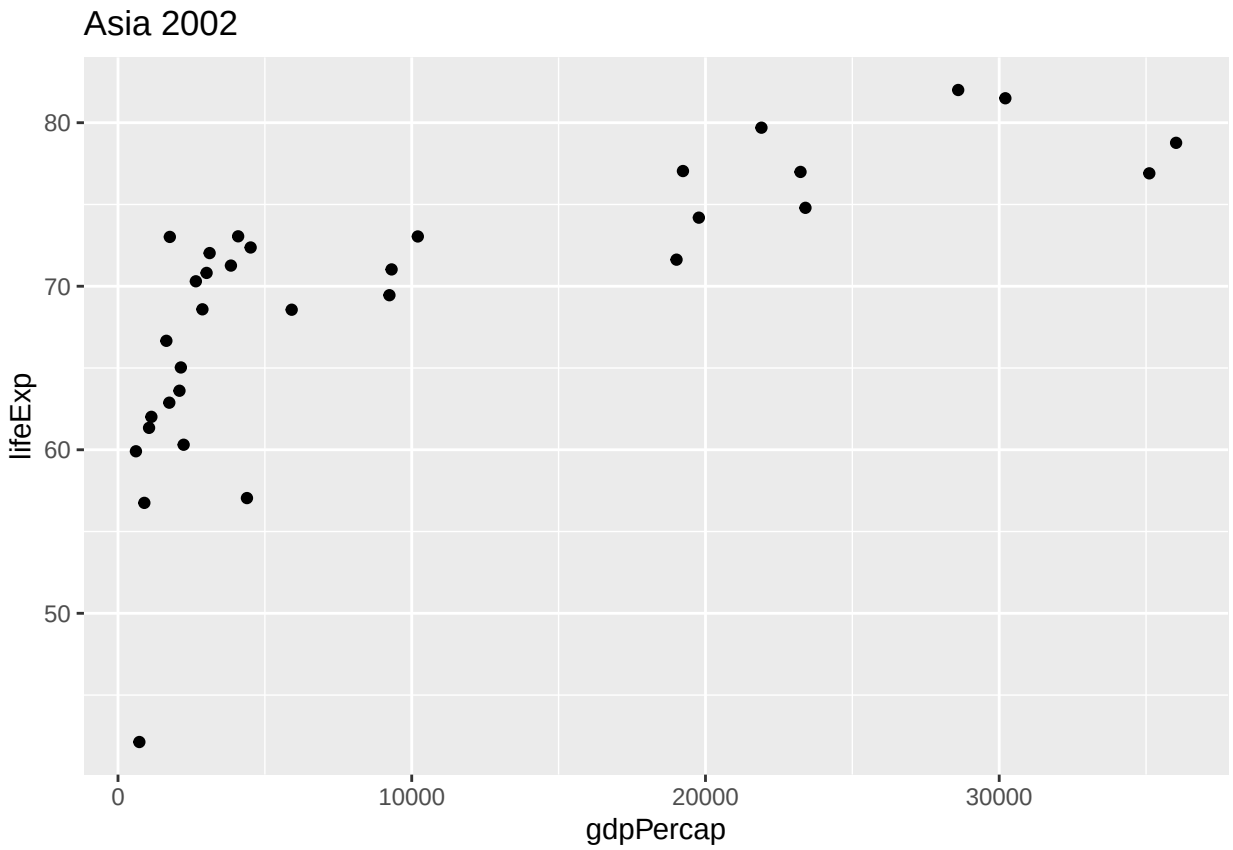

```
##
## [[9]]
```


Asia 1992
lifeExp
gdpPercap
80
70
60
50
40
0
10000
20000
30000

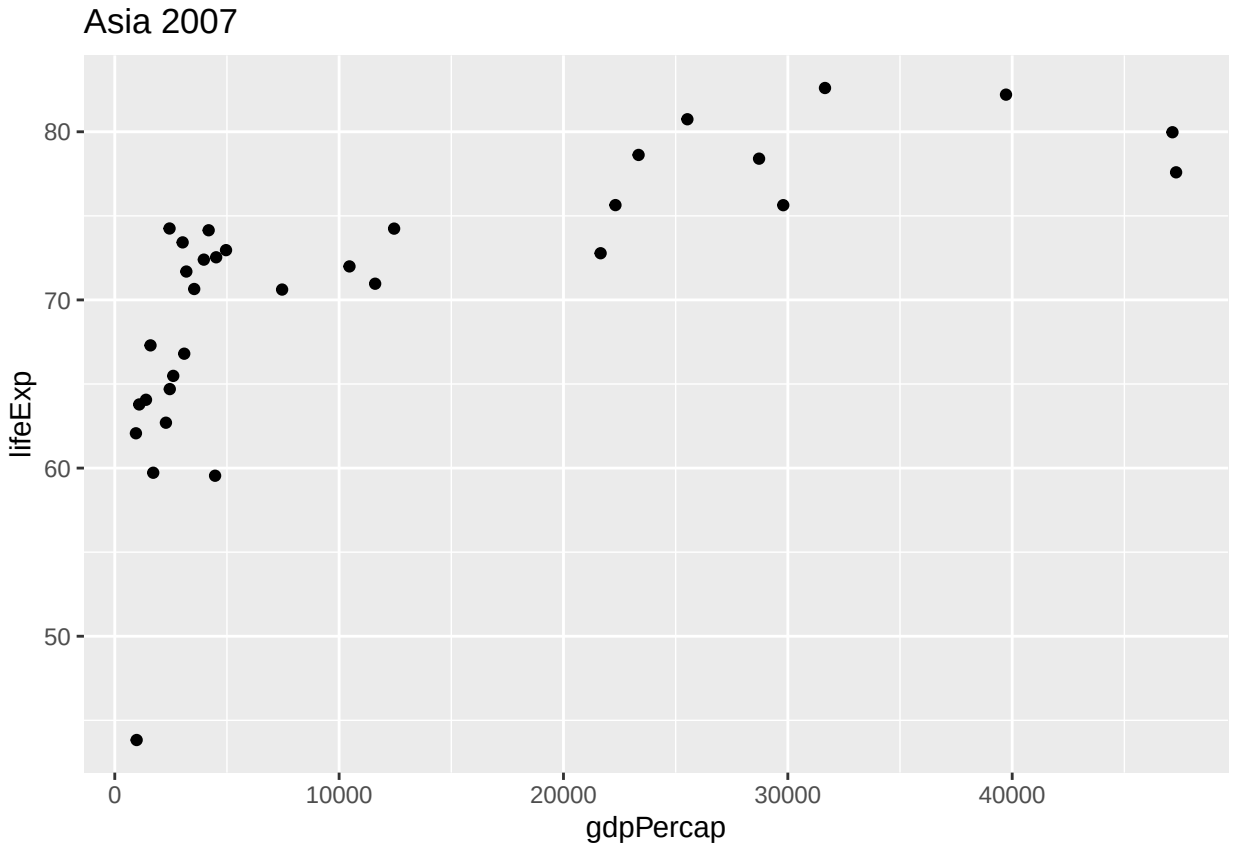
```
##
## [[10]]
```


Asia 1997
lifeExp
gdpPercap
80
70
60
50
40
0
10000
20000
30000
40000

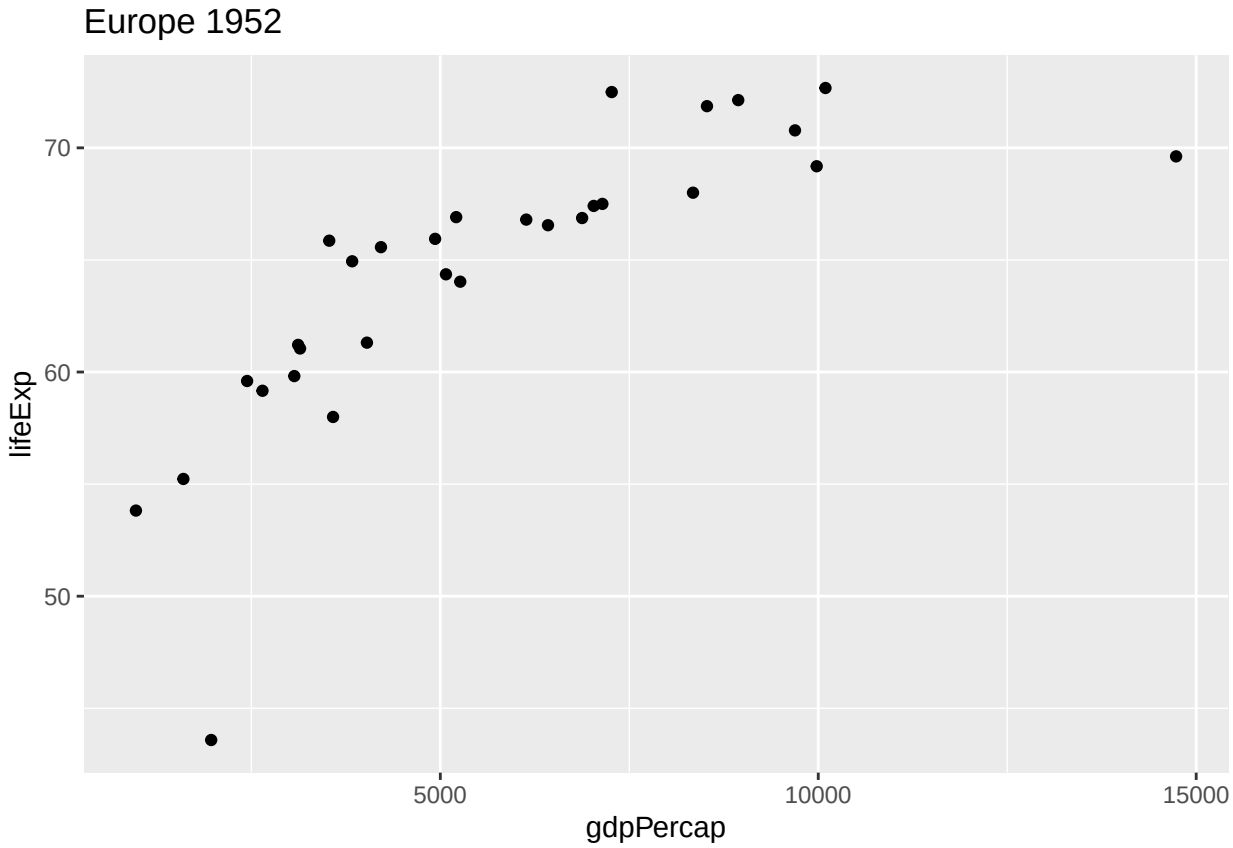
```
##
## [[11]]
```



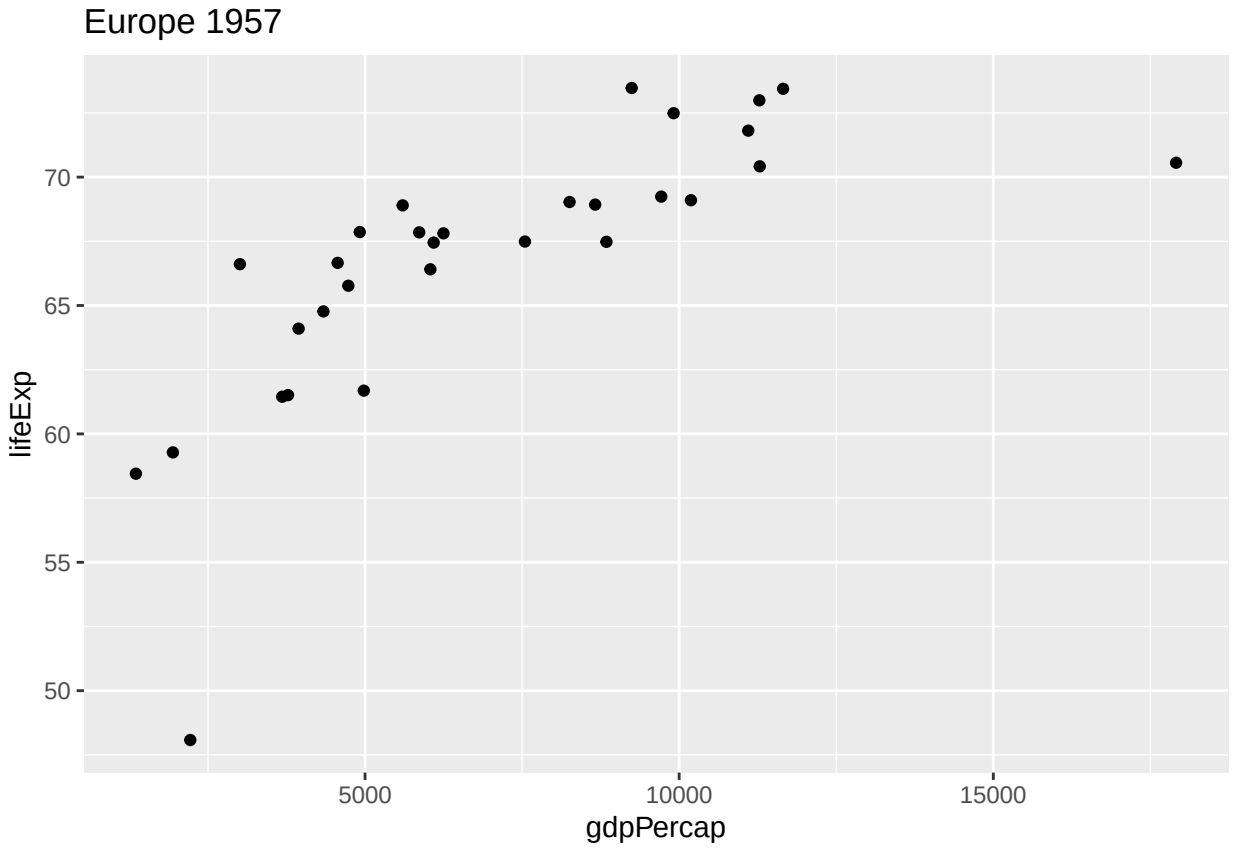

```
##
## [[12]]
```


Asia 2007
lifeExp
gdpPercap
80
70
60
50
0
10000
20000
30000
40000

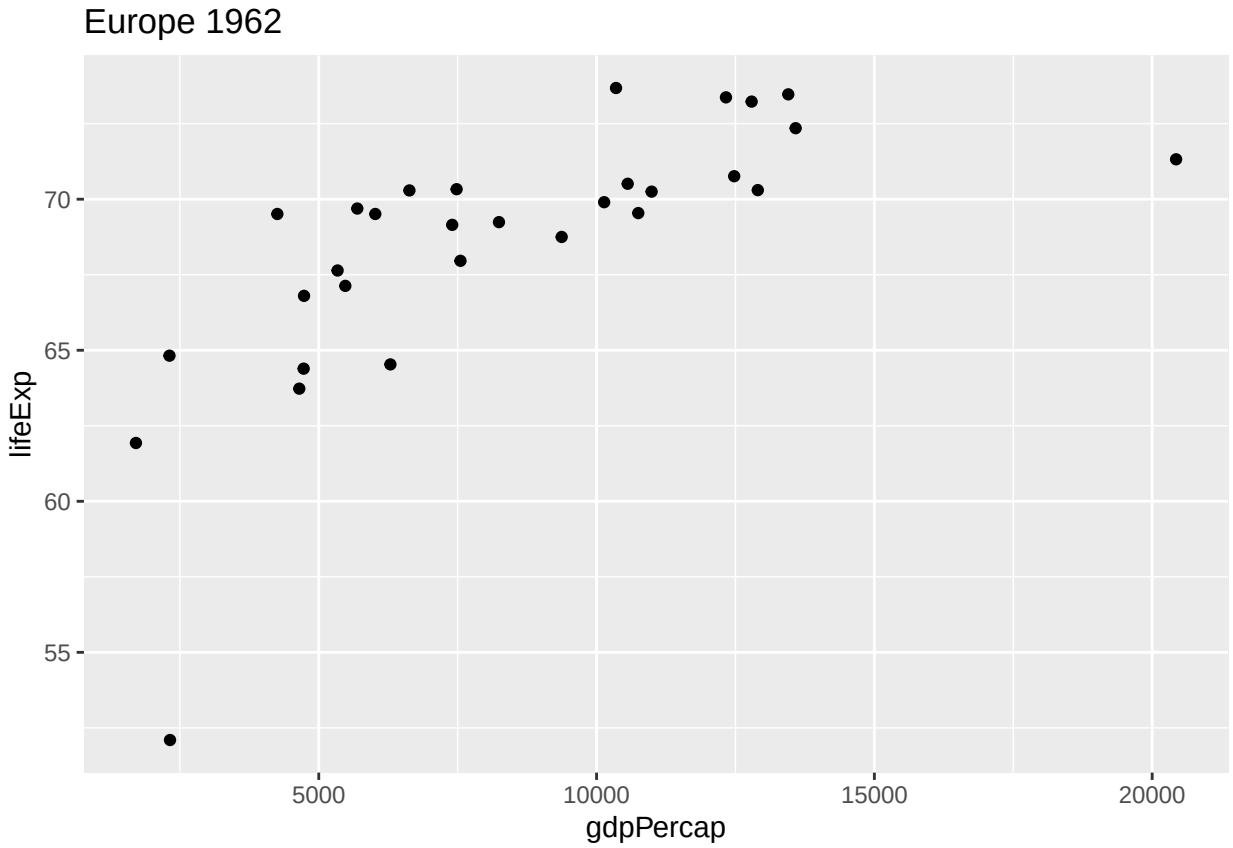
```
##
## [[13]]
```



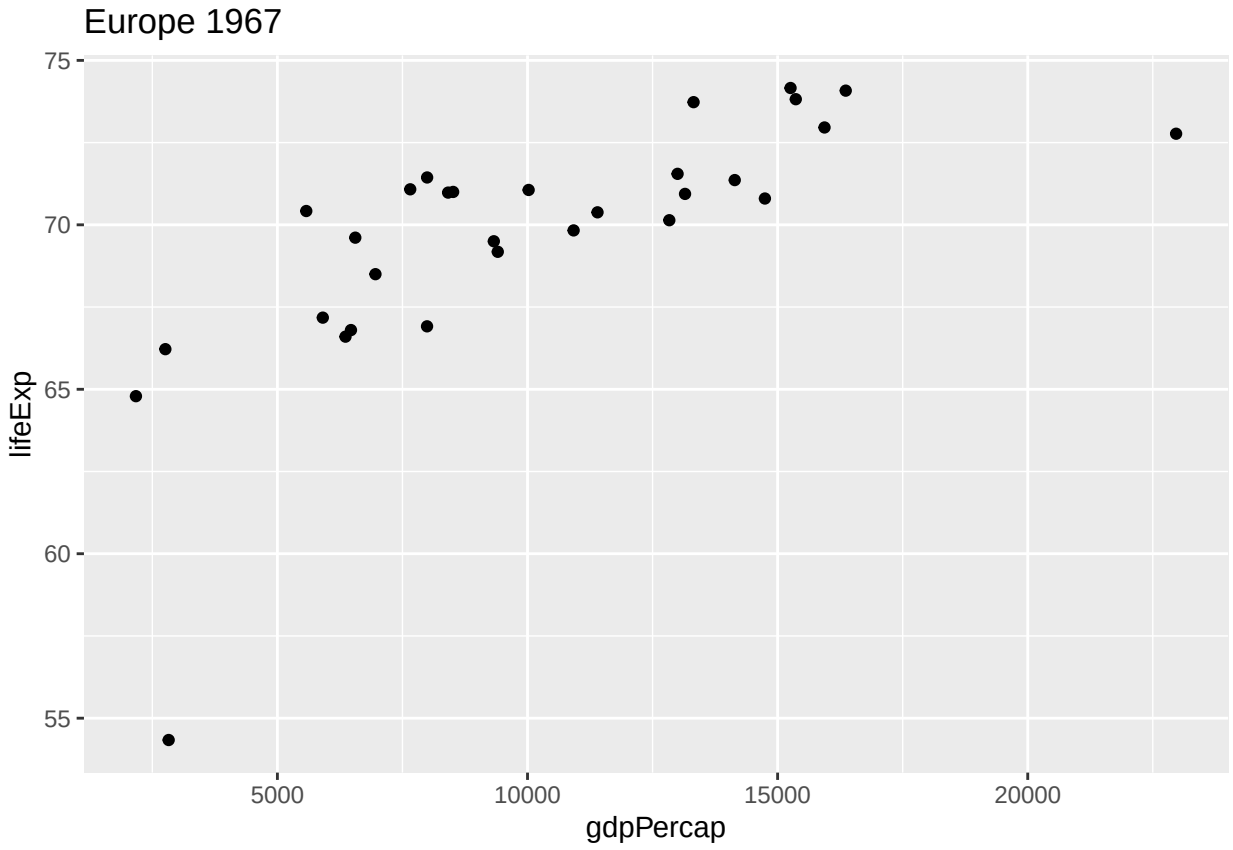

```
##
## [[14]]
```


Europe 1957
lifeExp
70
65
60
55
50
5000
10000
15000
gdpPercap

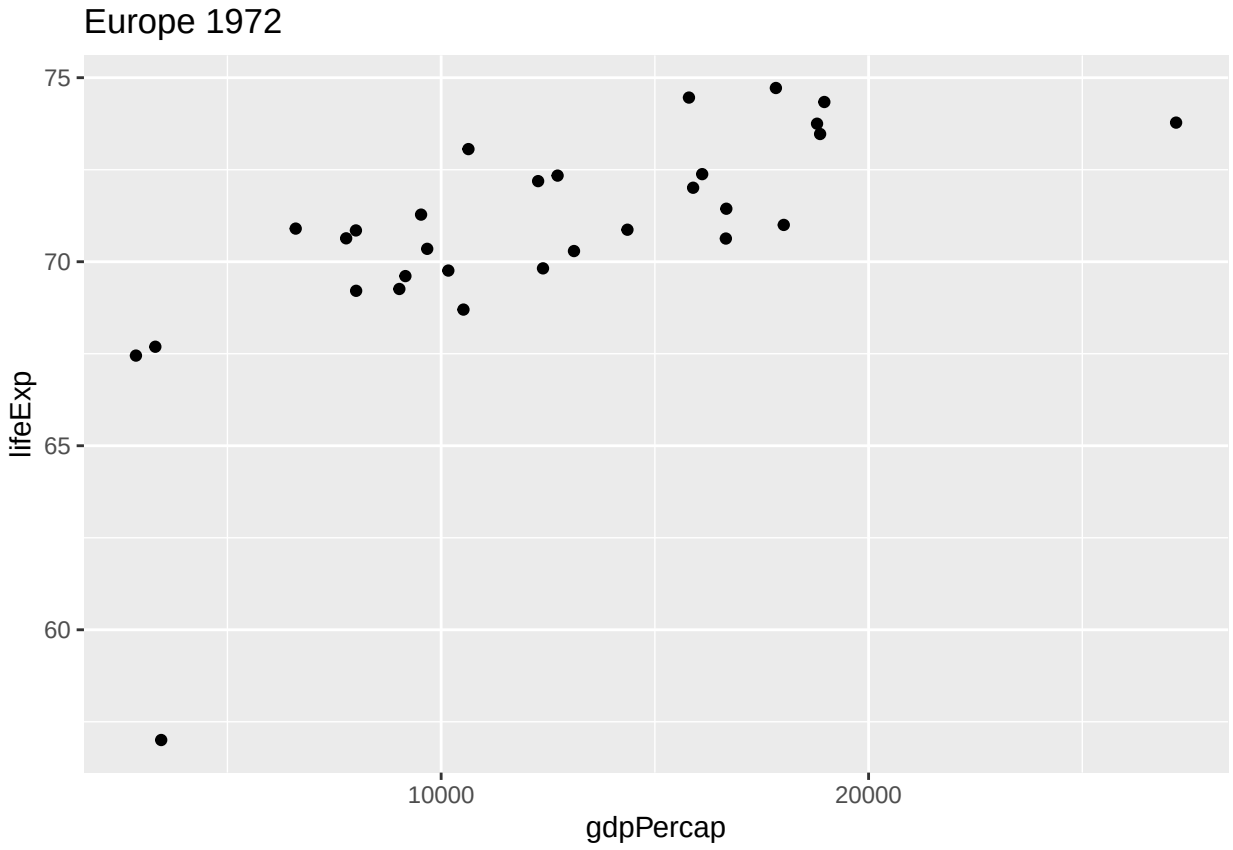
```
##
## [[15]]
```


Europe 1962
lifeExp
70
65
60
55
5000
10000
15000
20000
gdpPercap

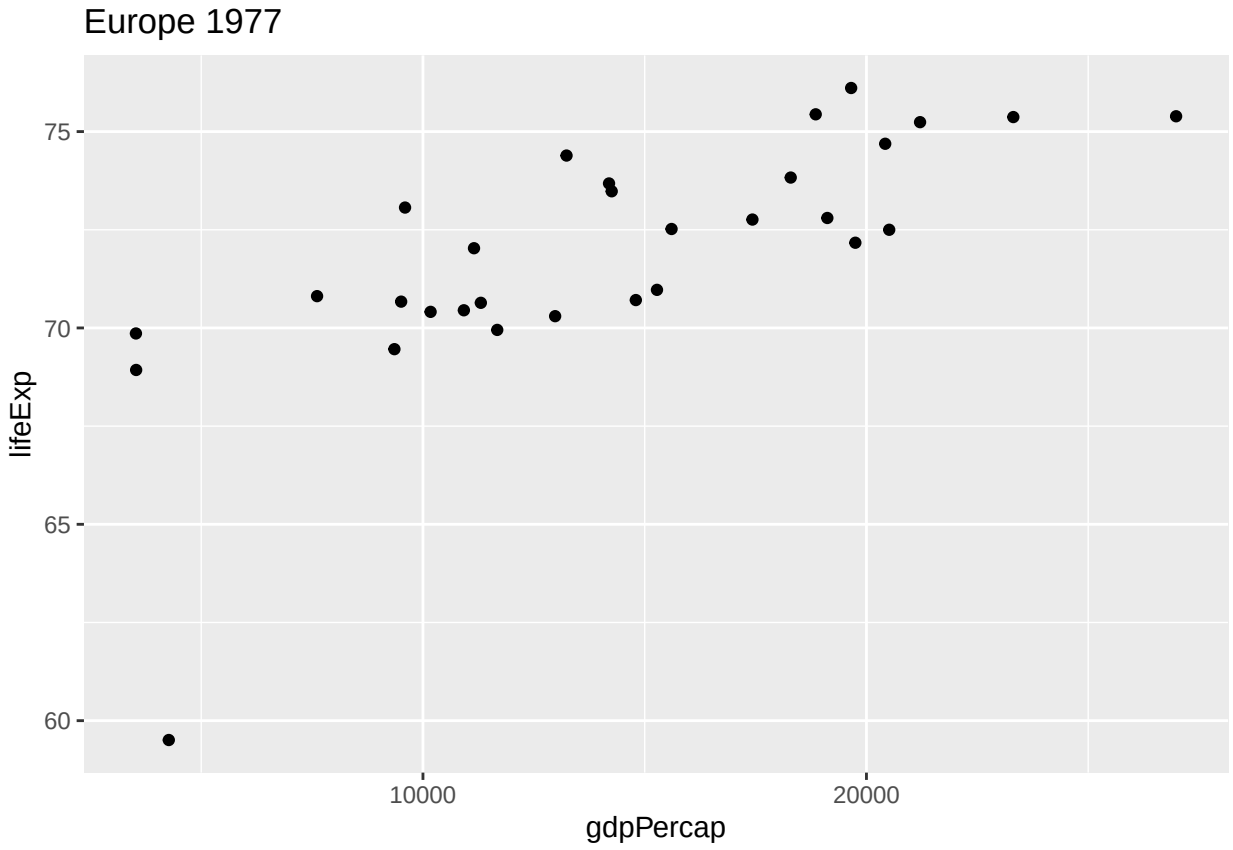
```
##
## [[16]]
```


Europe 1967
lifeExp
gdpPercap
75
70
65
60
55
5000
10000
15000
20000

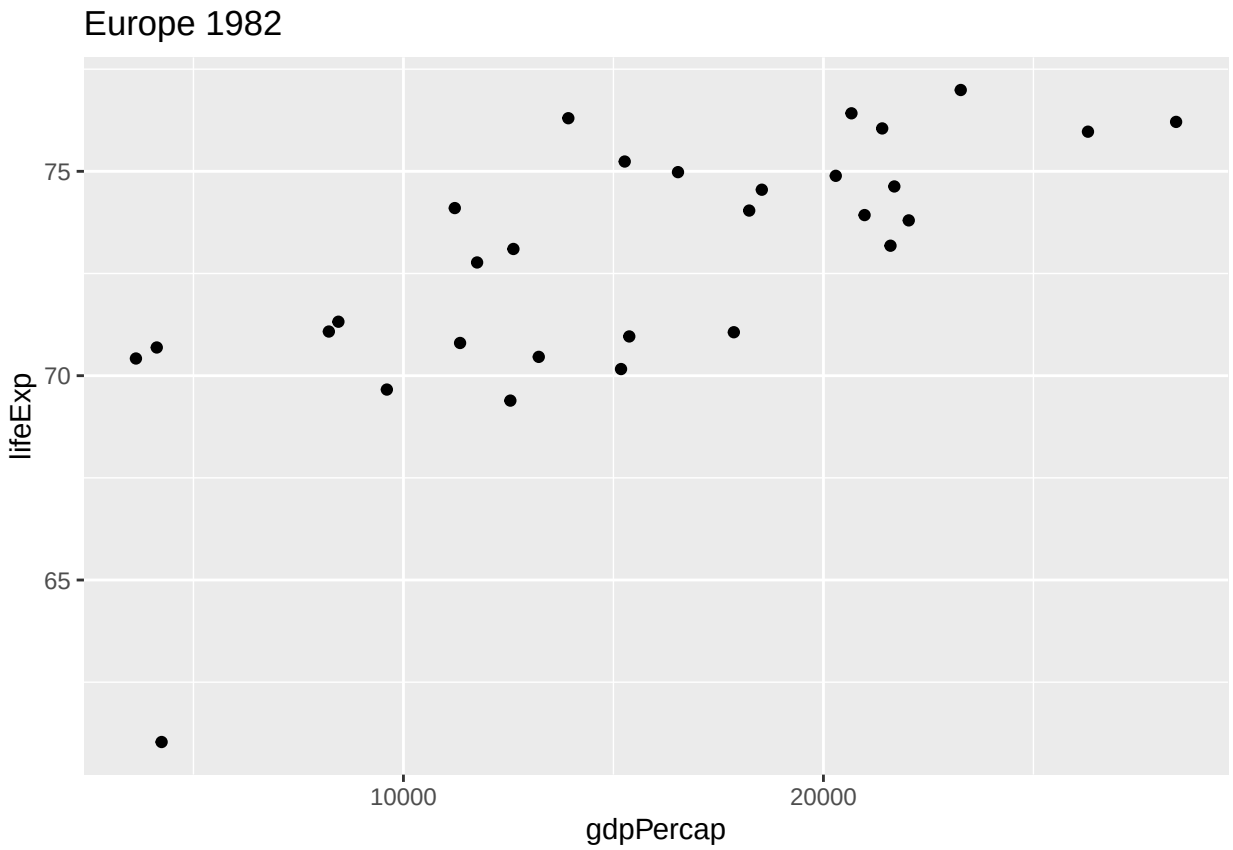
```
##
## [[17]]
```



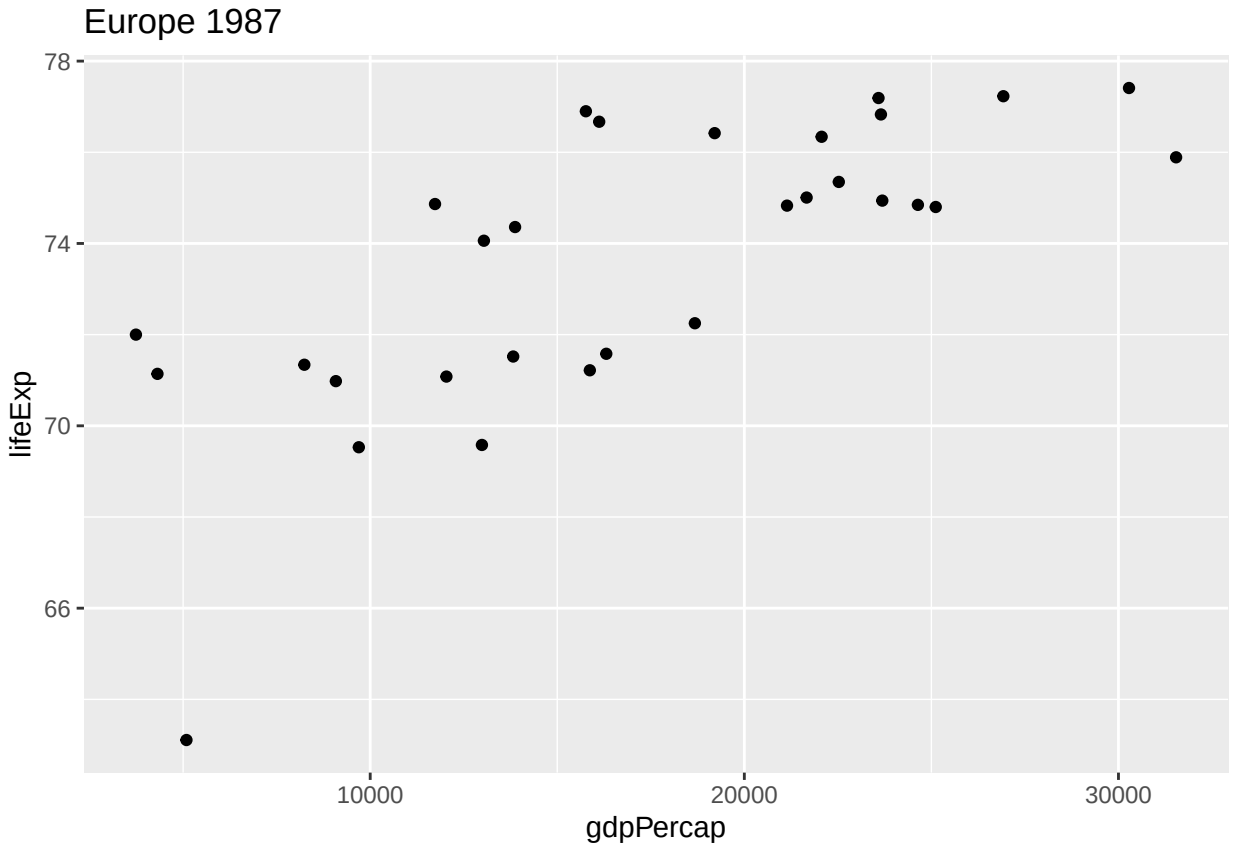

```
##
## [[18]]
```



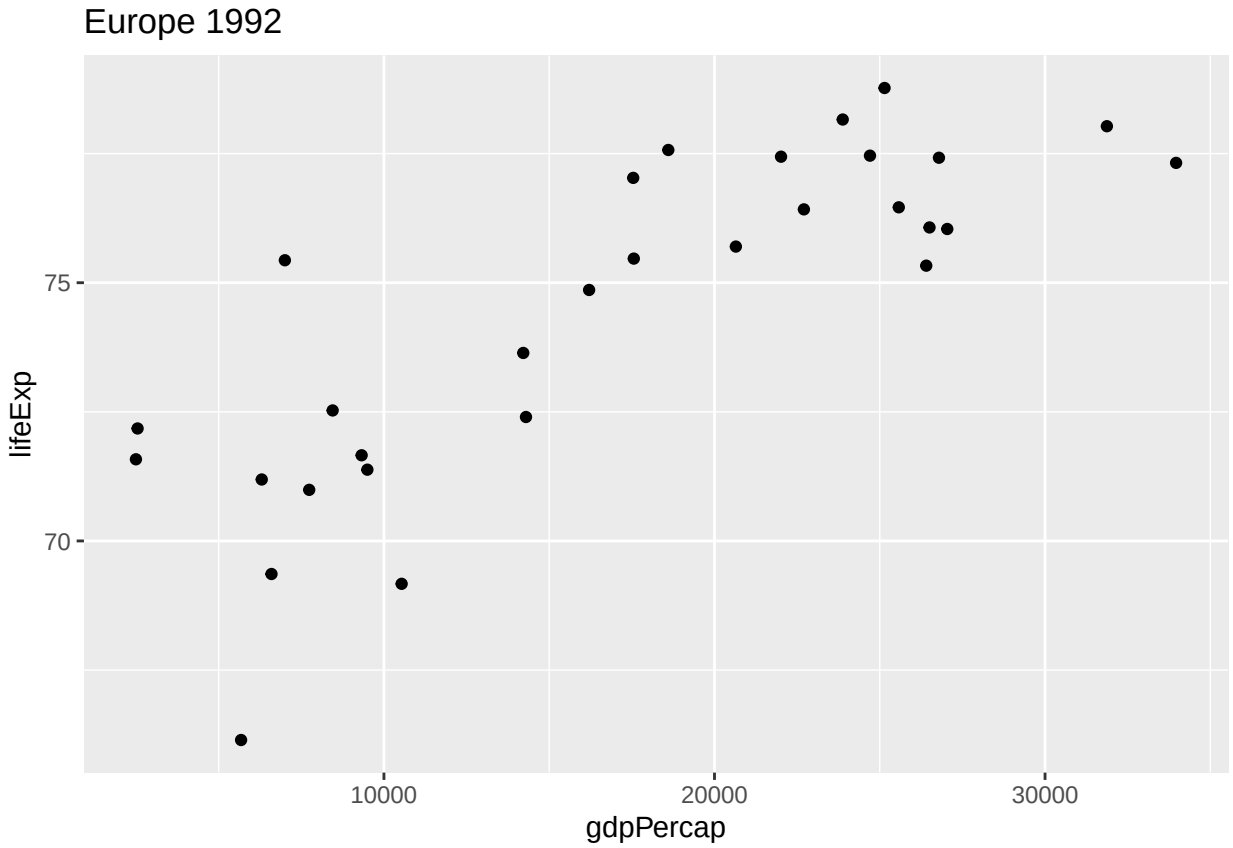

```
##
## [[19]]
```



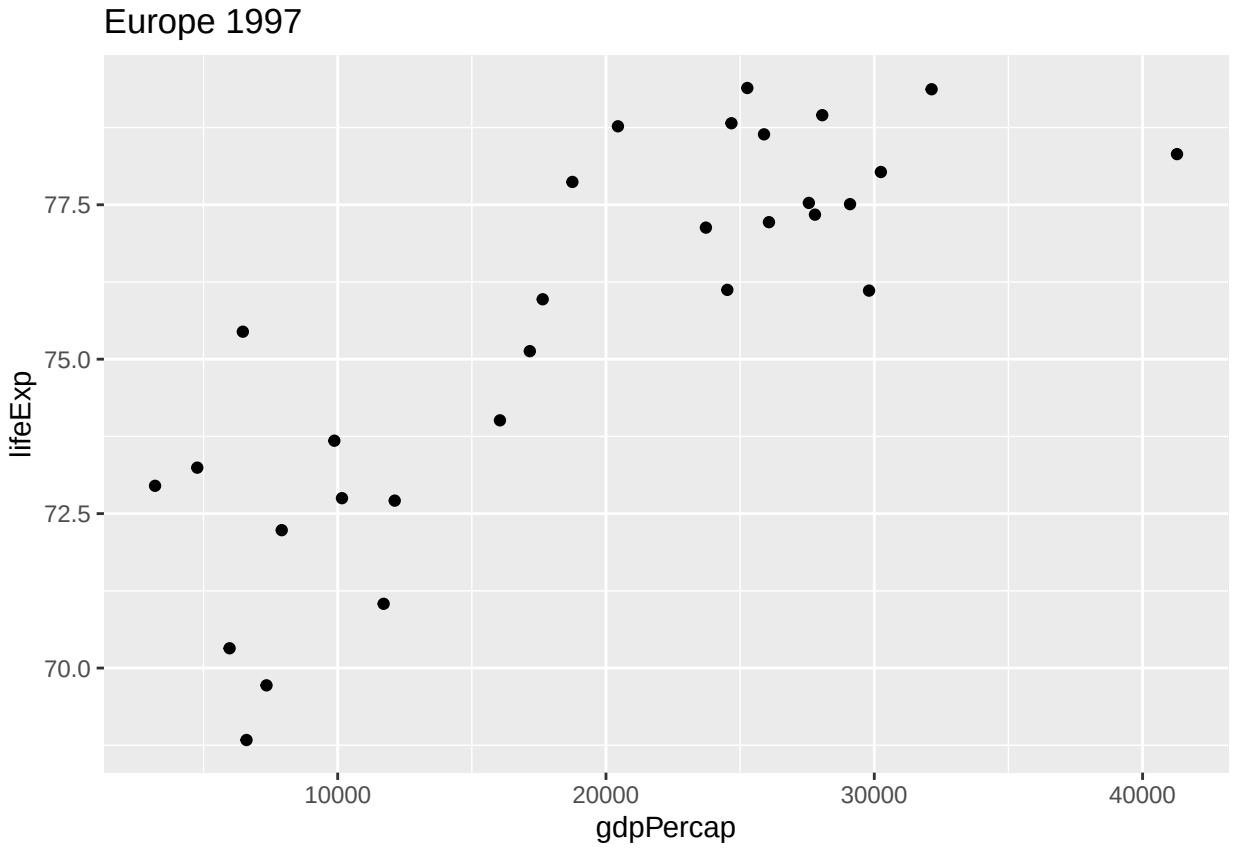

```
##
## [[20]]
```



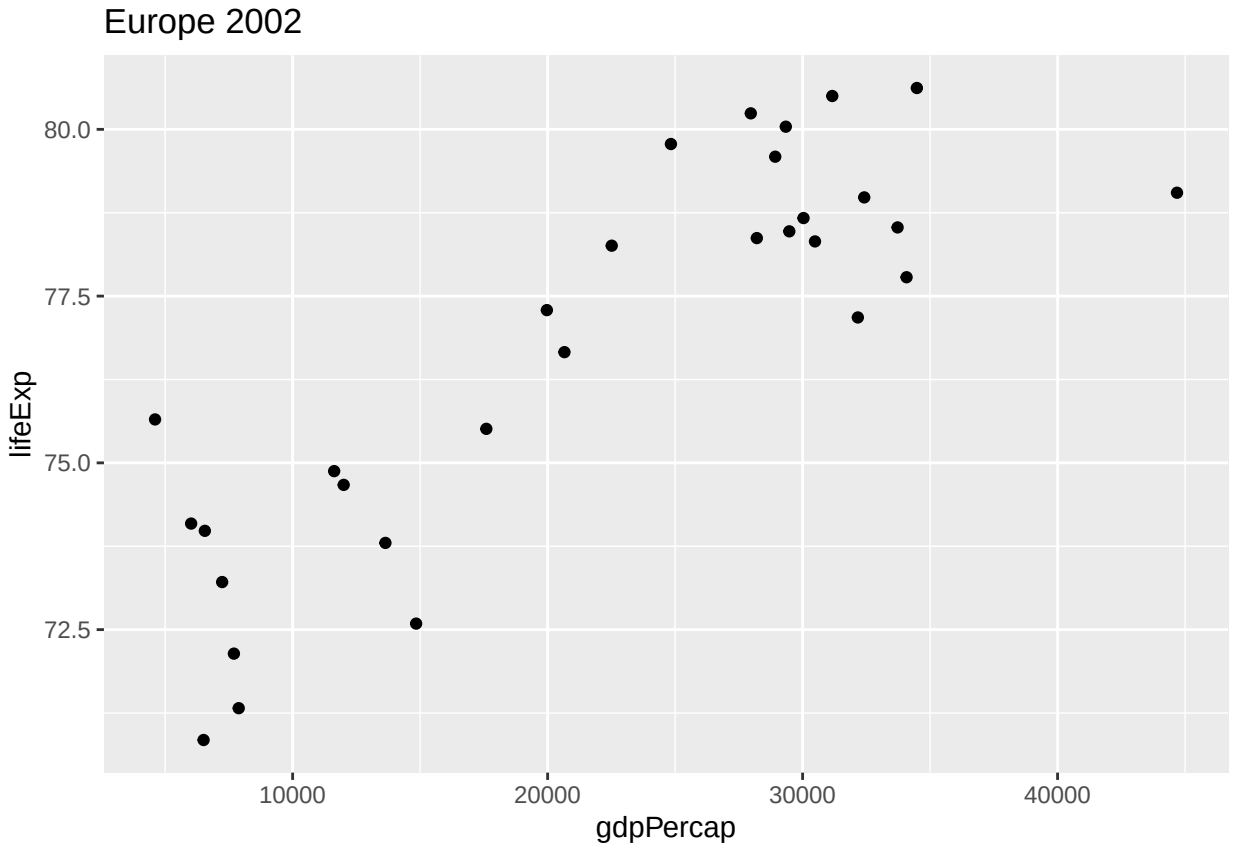

```
##
## [[21]]
```



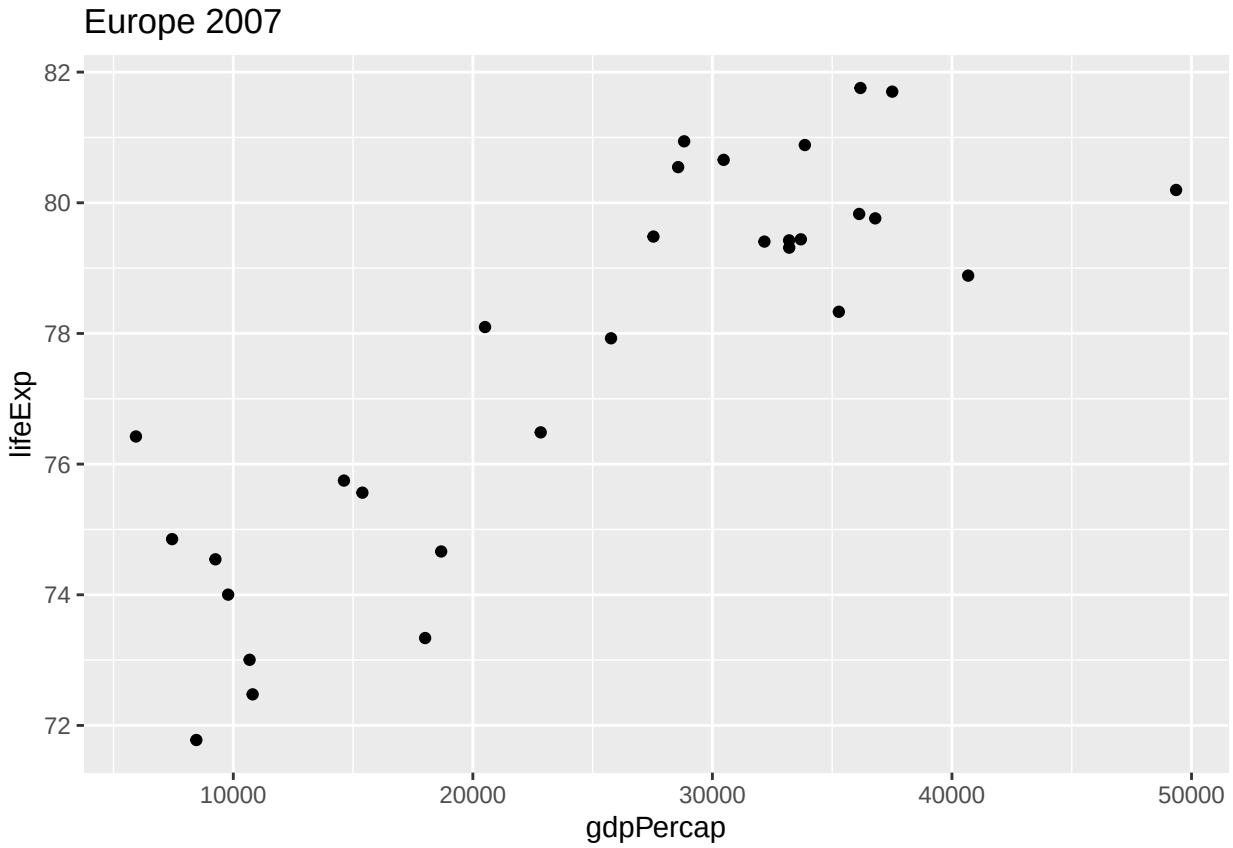

```
##
## [[22]]
```


Europe 1997
lifeExp
77.5
75.0
72.5
70.0
10000
20000
30000
40000
gdpPercap

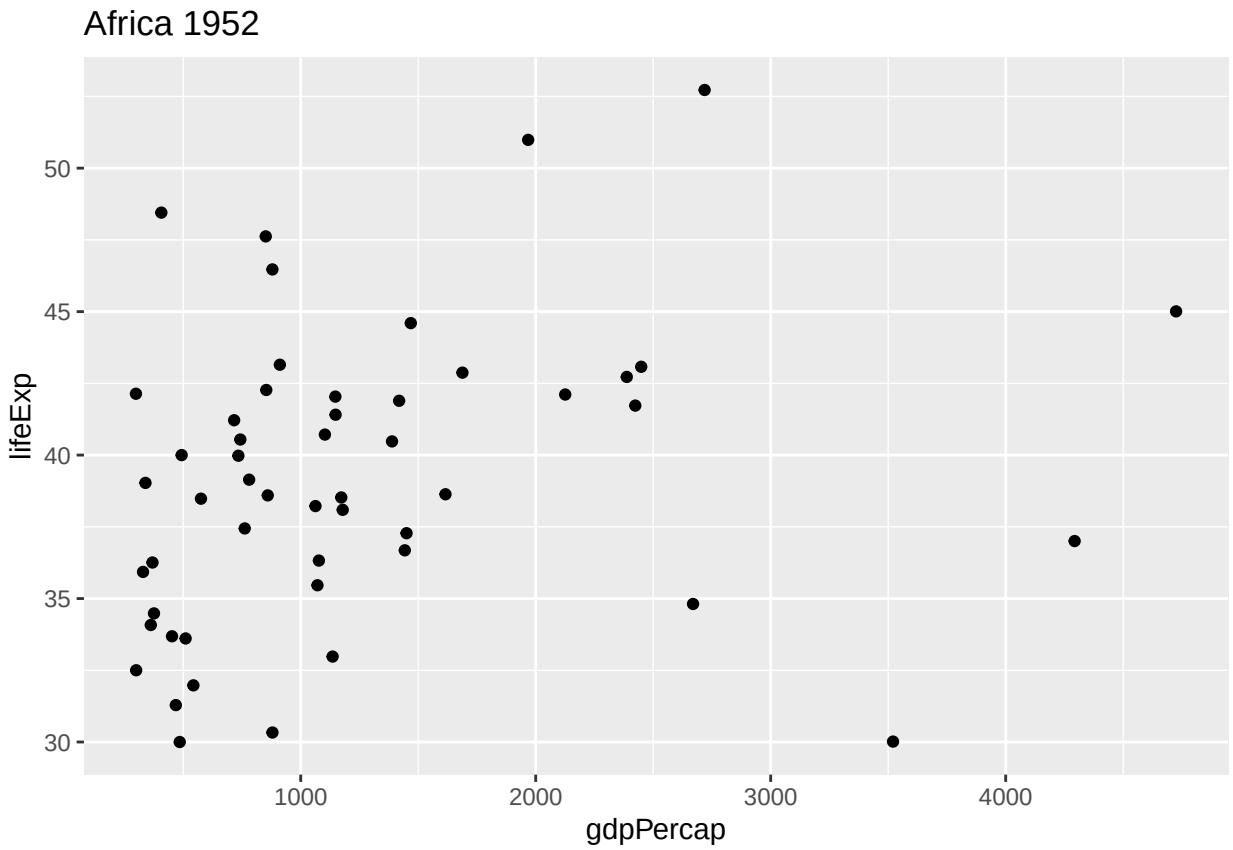
```
##
## [[23]]
```


Europe 2002
lifeExp
80.0
77.5
75.0
72.5
10000
20000
30000
40000
gdpPercap

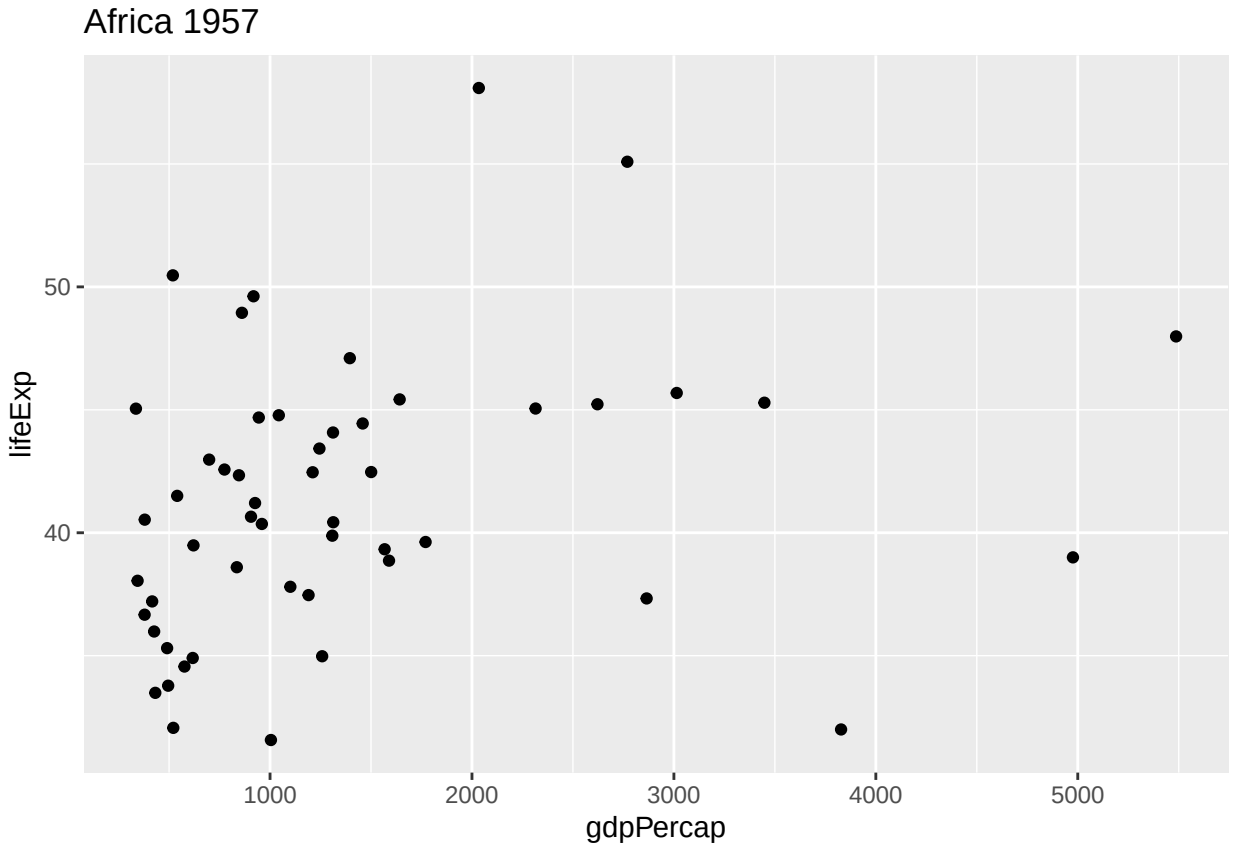
```
##
## [[24]]
```


Europe 2007
lifeExp
gdpPercap
82
80
78
76
74
72
10000
20000
30000
40000
50000

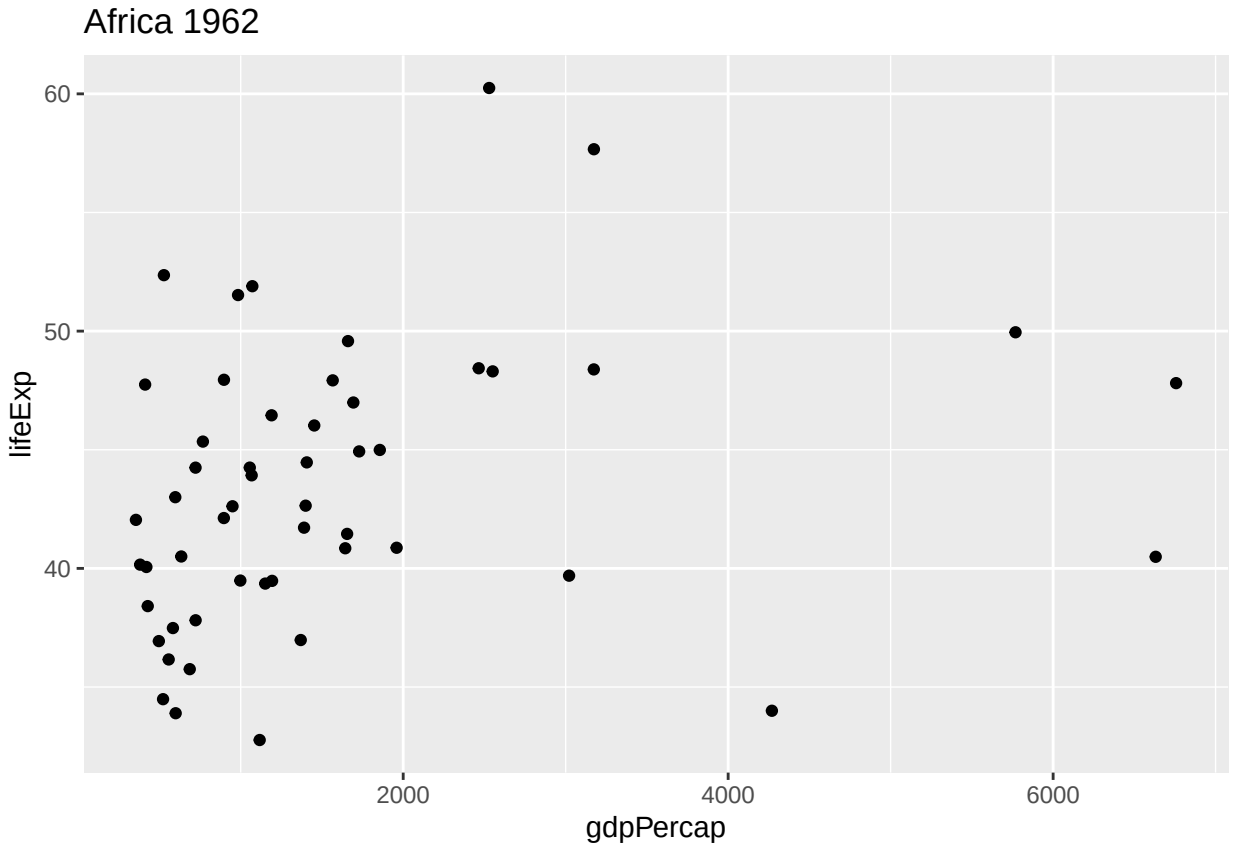
```
##
## [[25]]
```


Africa 1952
lifeExp
50
45
40
35
30
1000
2000
3000
4000
gdpPercap

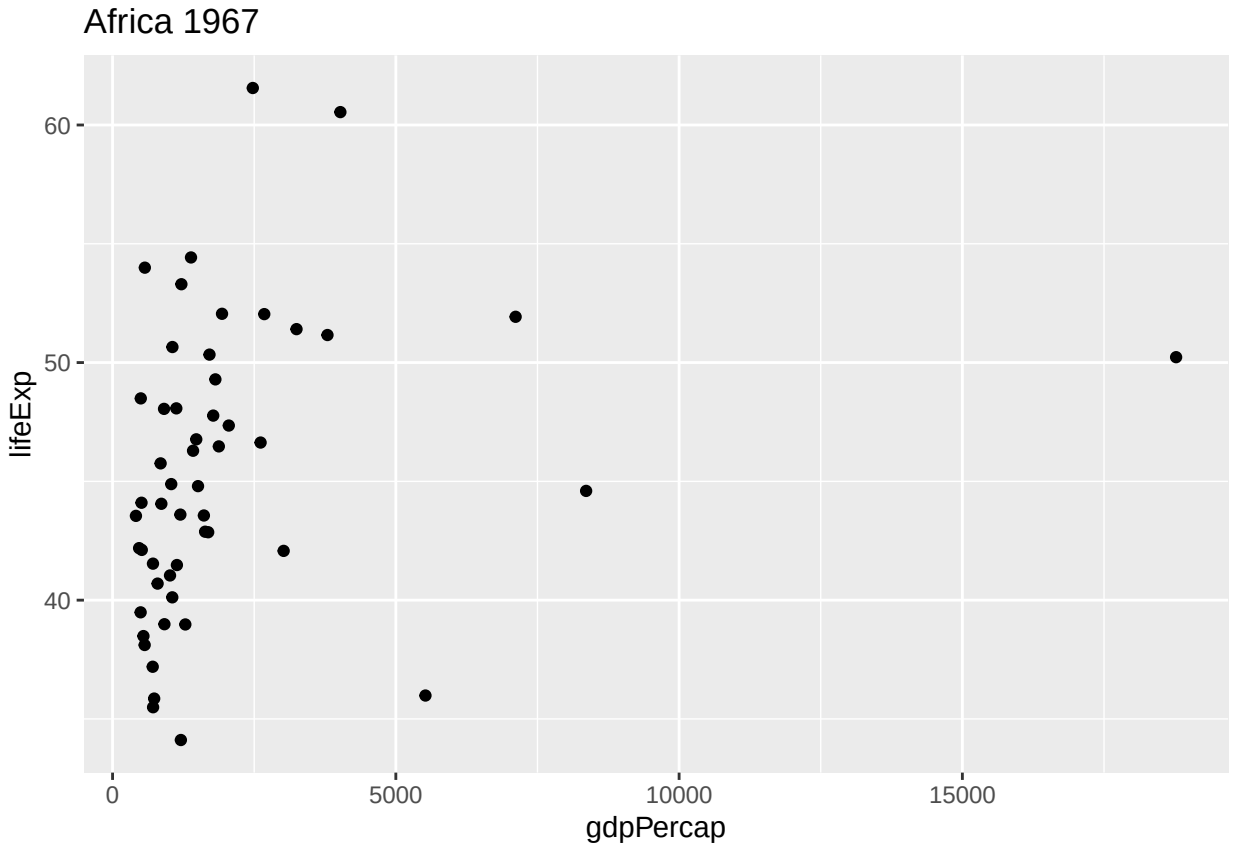
```
##
## [[26]]
```


Africa 1957
lifeExp
50
40
1000
2000
3000
4000
5000
gdpPercap

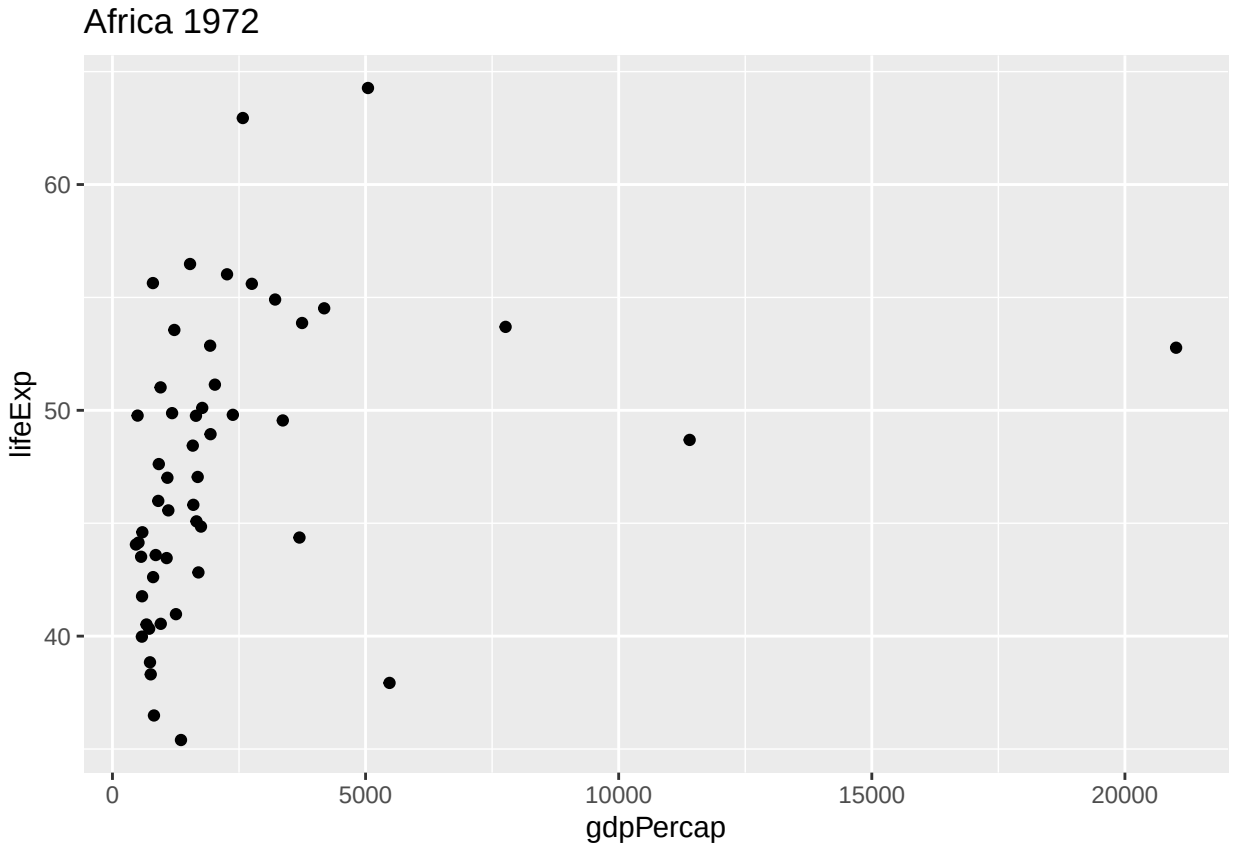
```
##
## [[27]]
```



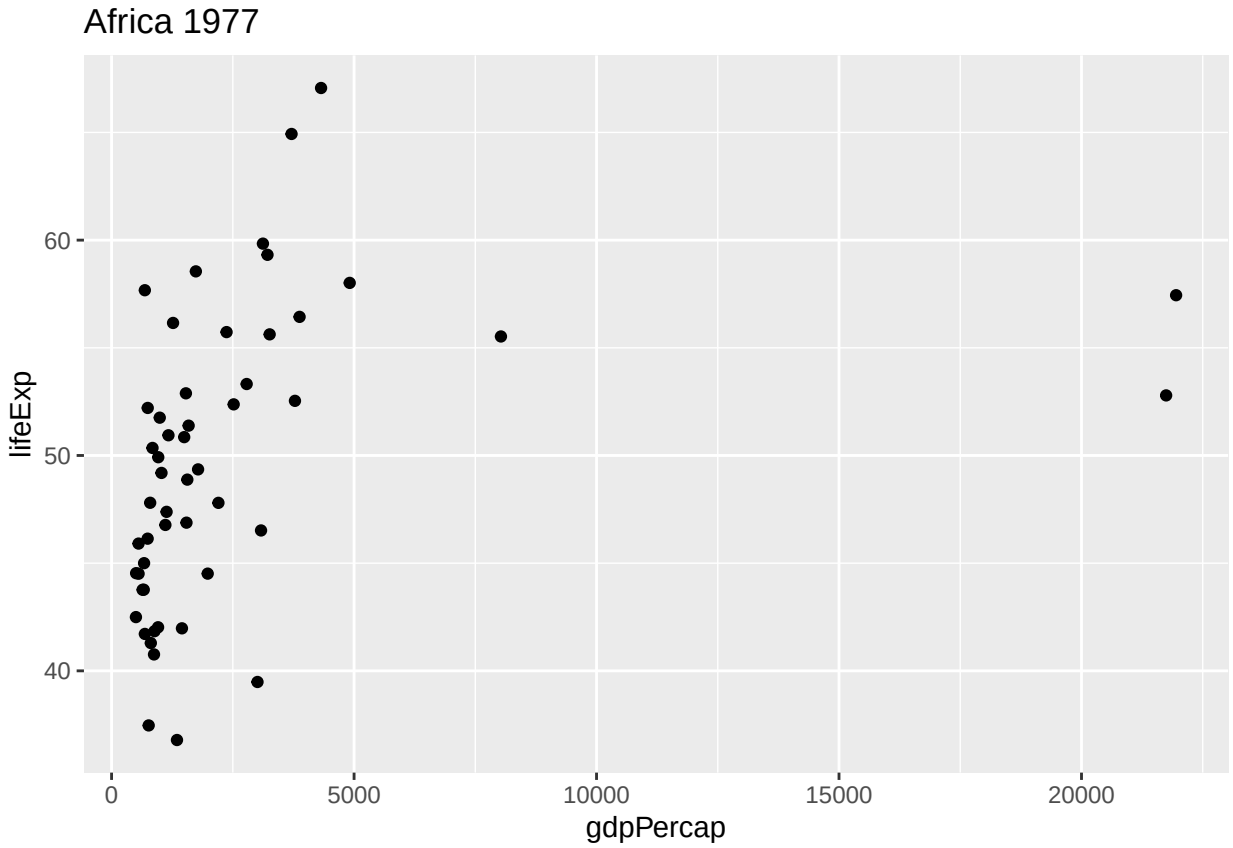

```
##
## [[28]]
```



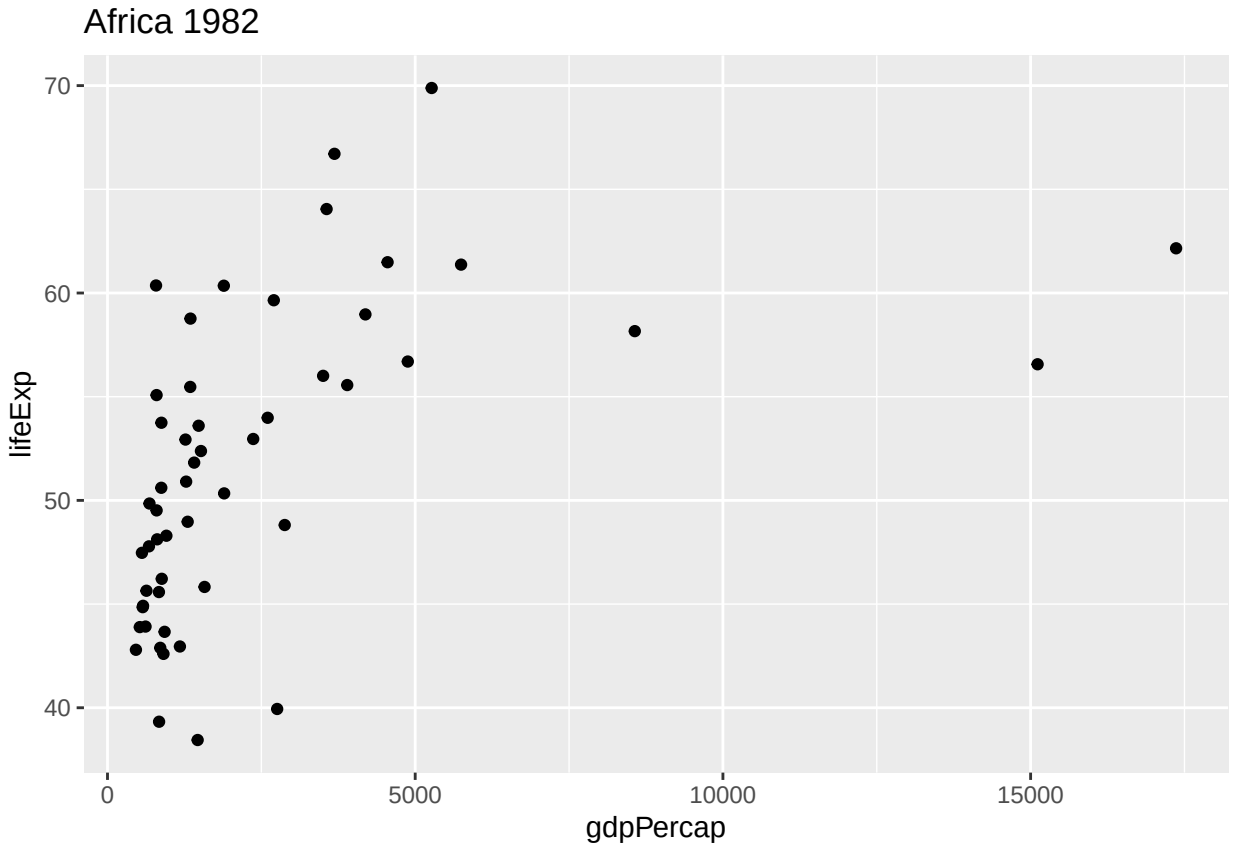

```
##
## [[29]]
```


Africa 1972
lifeExp
60
50
40
0
5000
10000
15000
20000
gdpPercap

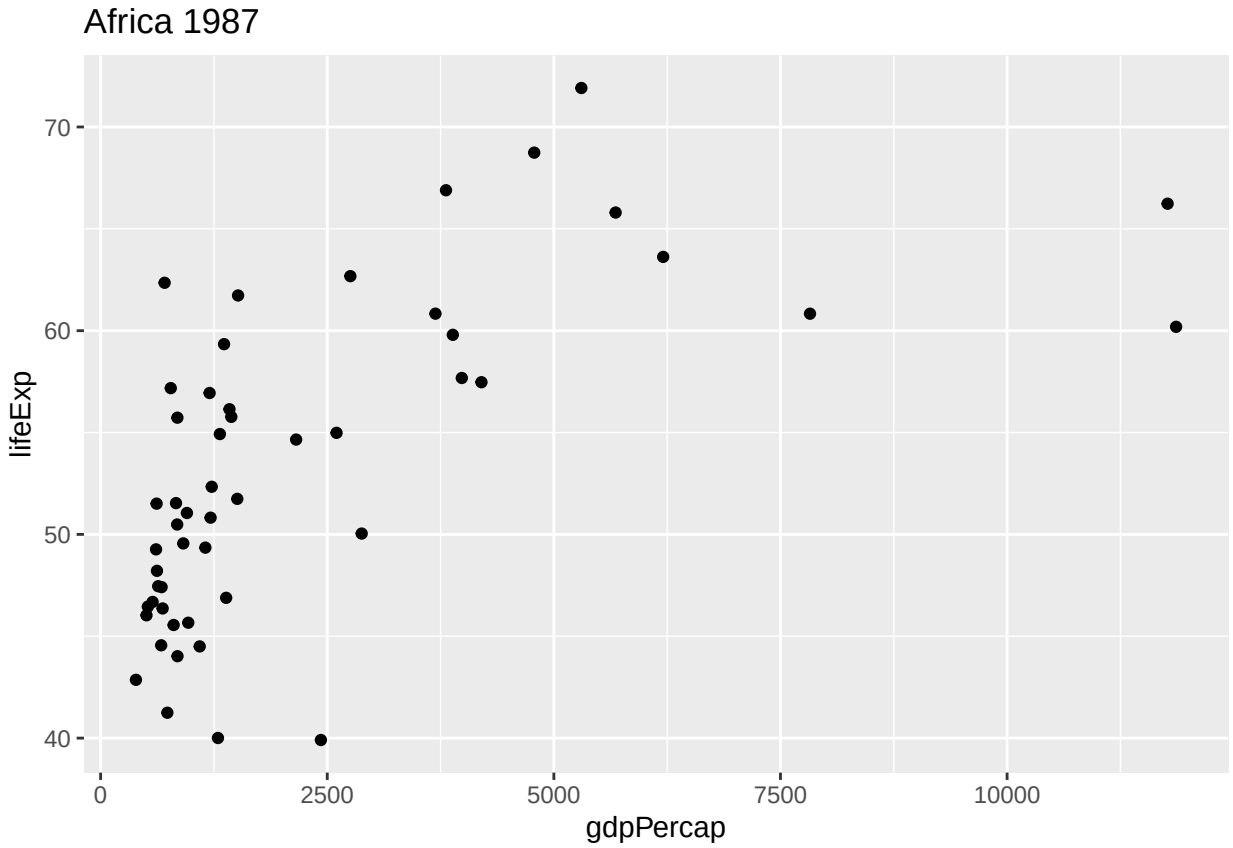
```
##
## [[30]]
```


Africa 1977
lifeExp
60
50
40
0
5000
10000
15000
20000
gdpPercap

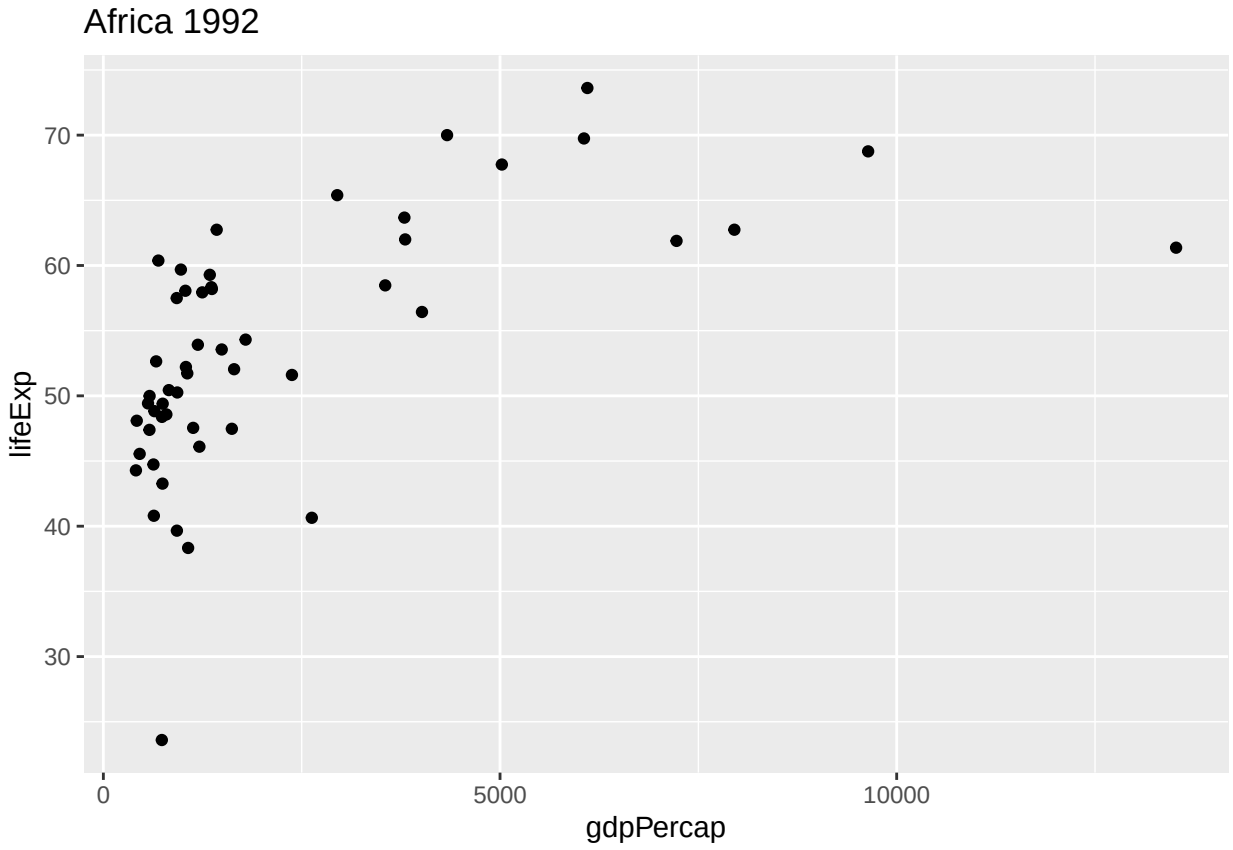
```
##
## [[31]]
```



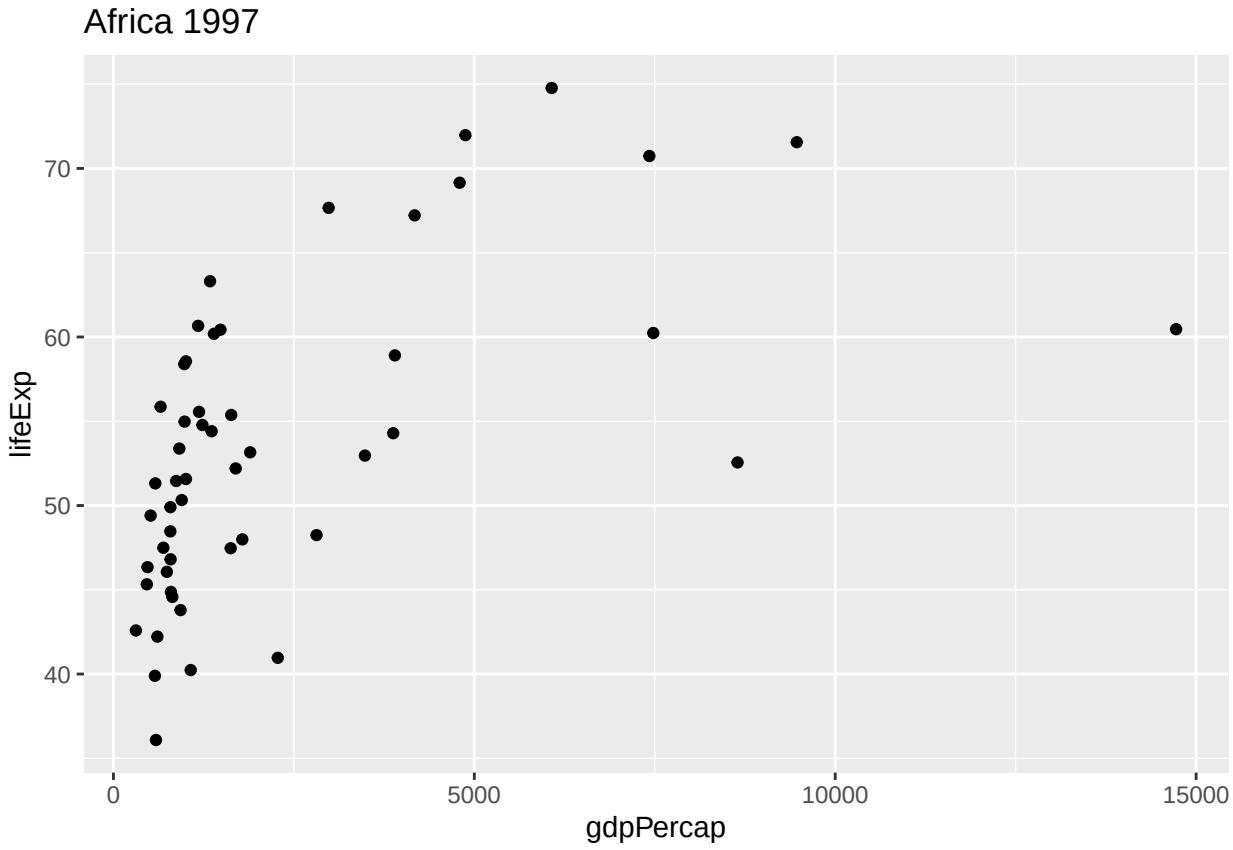

```
##
## [[32]]
```


Africa 1987
lifeExp
70
60
50
40
0
2500
5000
7500
10000
gdpPercap

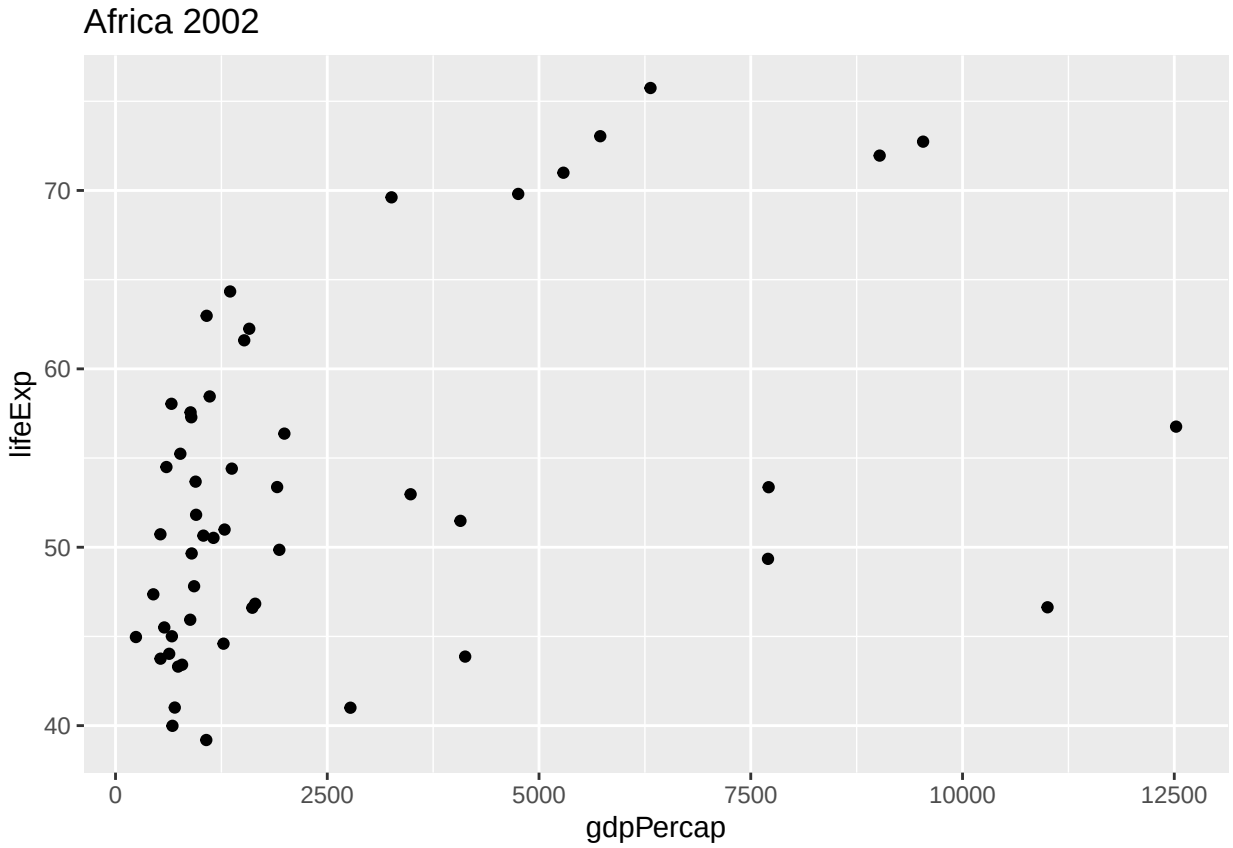
```
##
## [[33]]
```



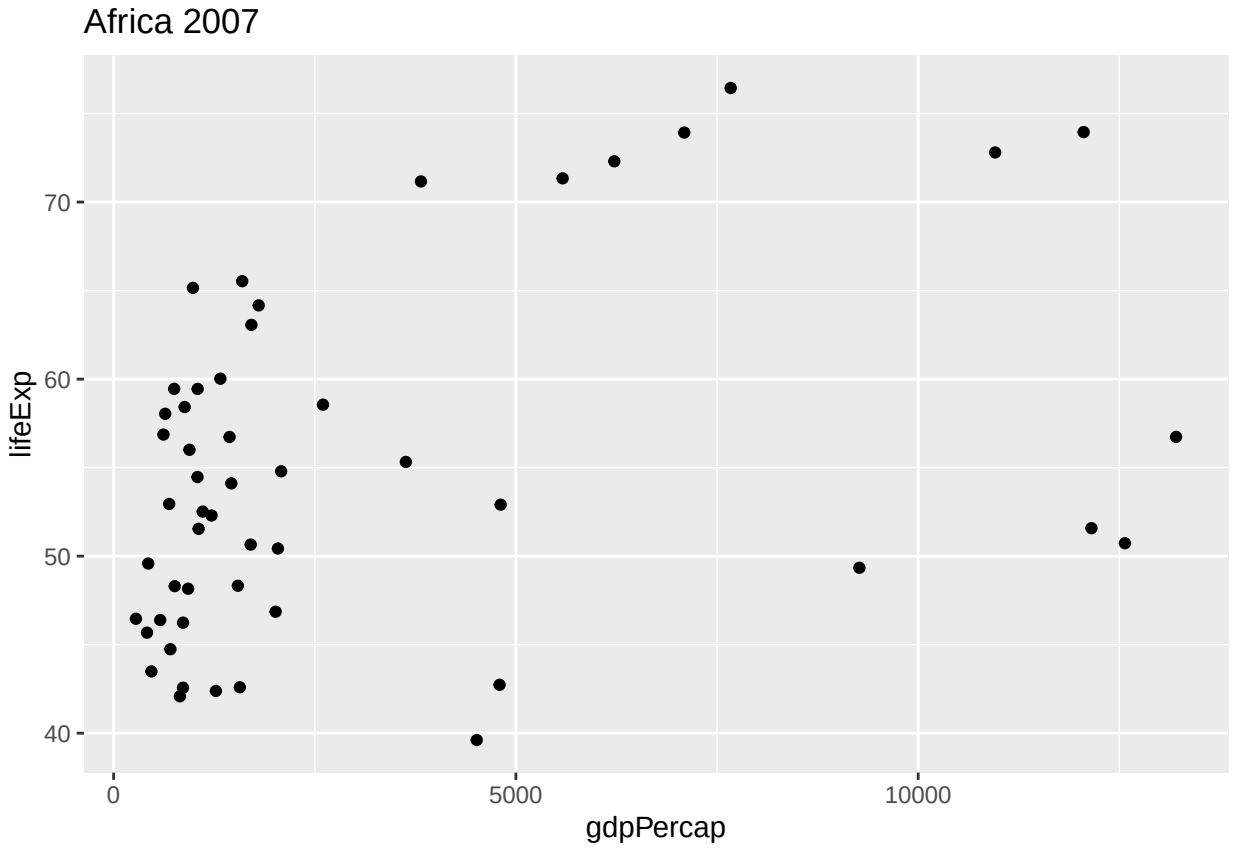

```
##
## [[34]]
```



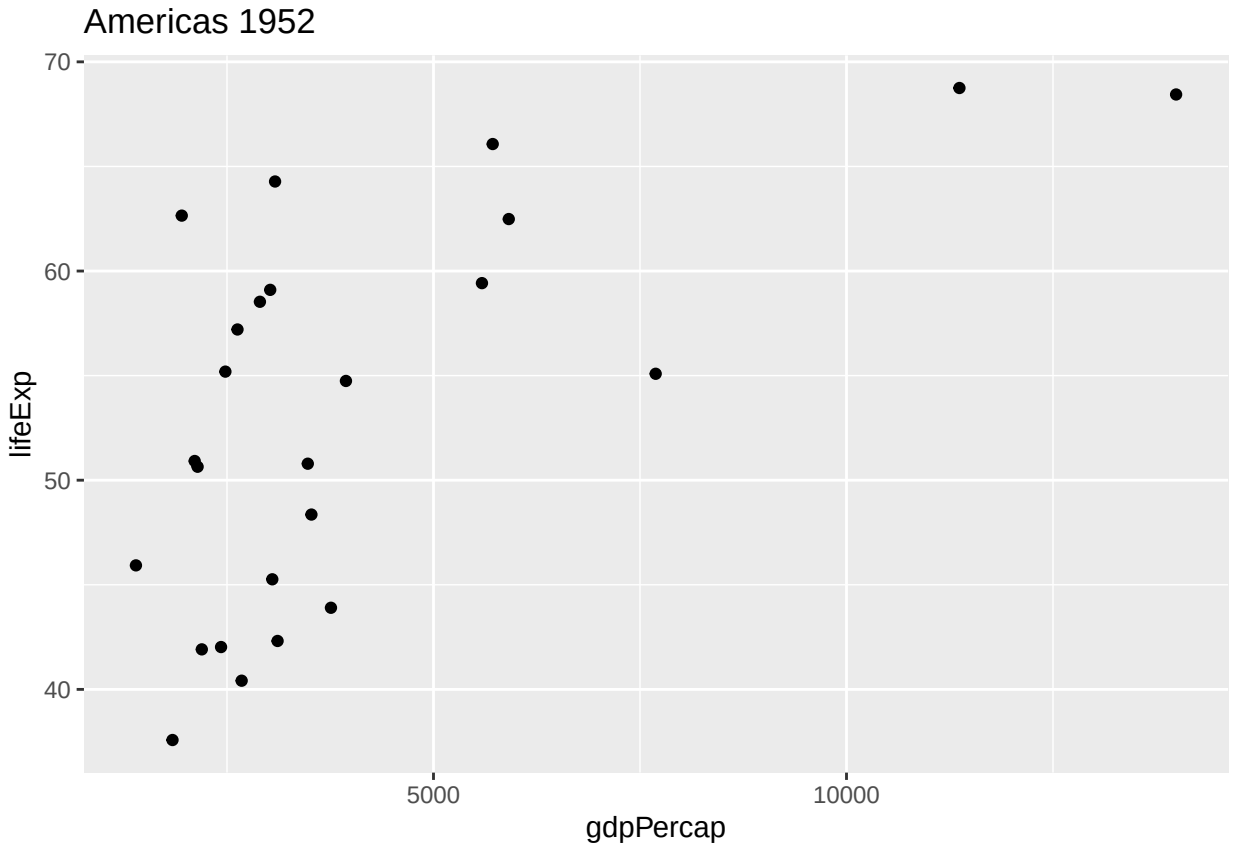

```
##
## [[35]]
```


Africa 2002
lifeExp
70
60
50
40
0
2500
5000
7500
10000
12500
gdpPercap

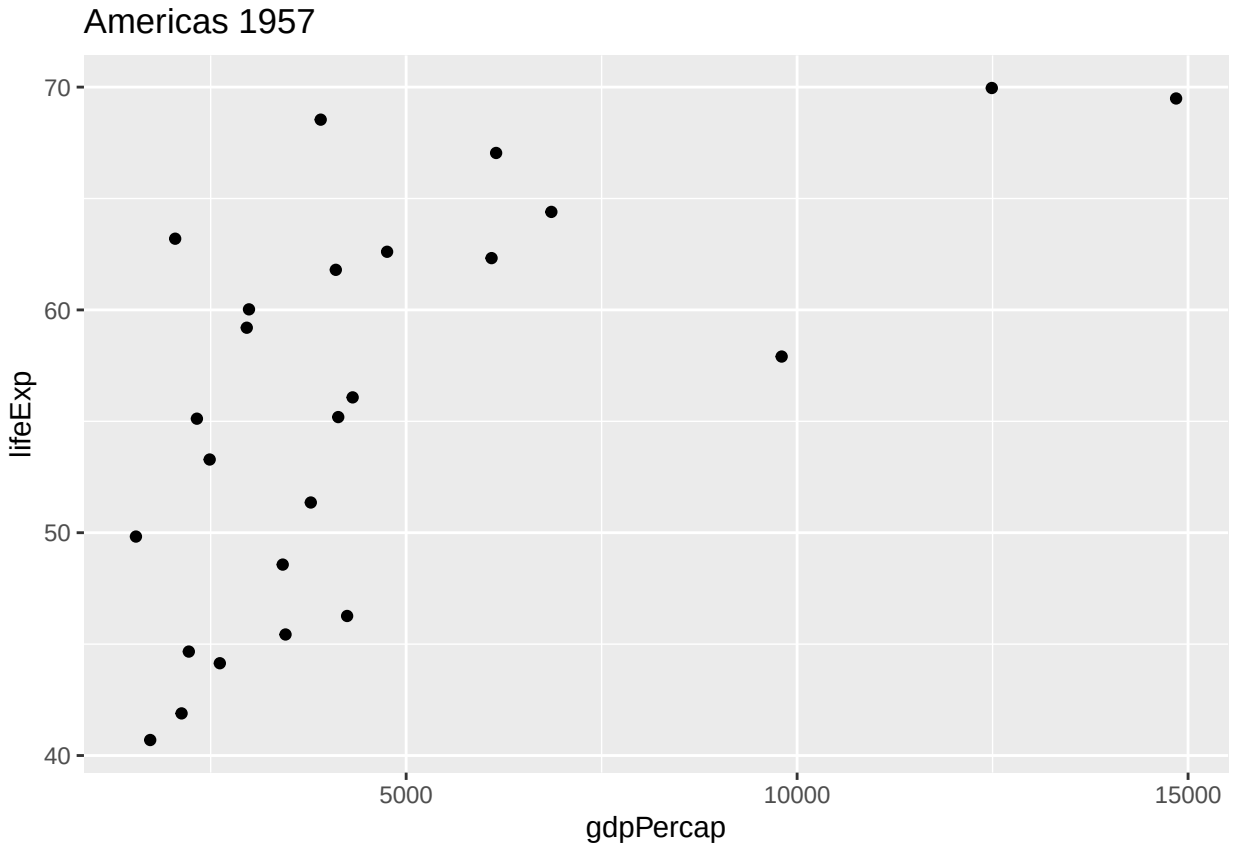
```
##
## [[36]]
```



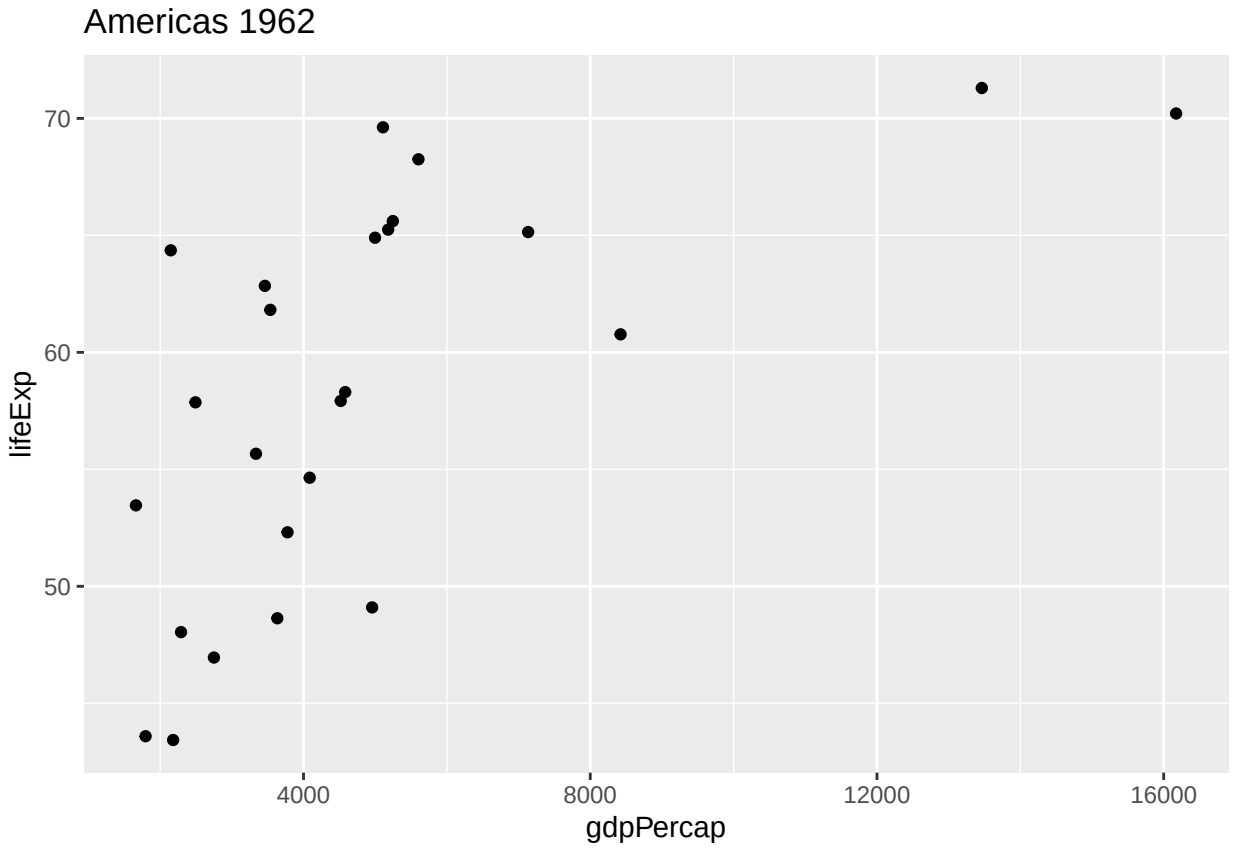

```
##
## [[37]]
```



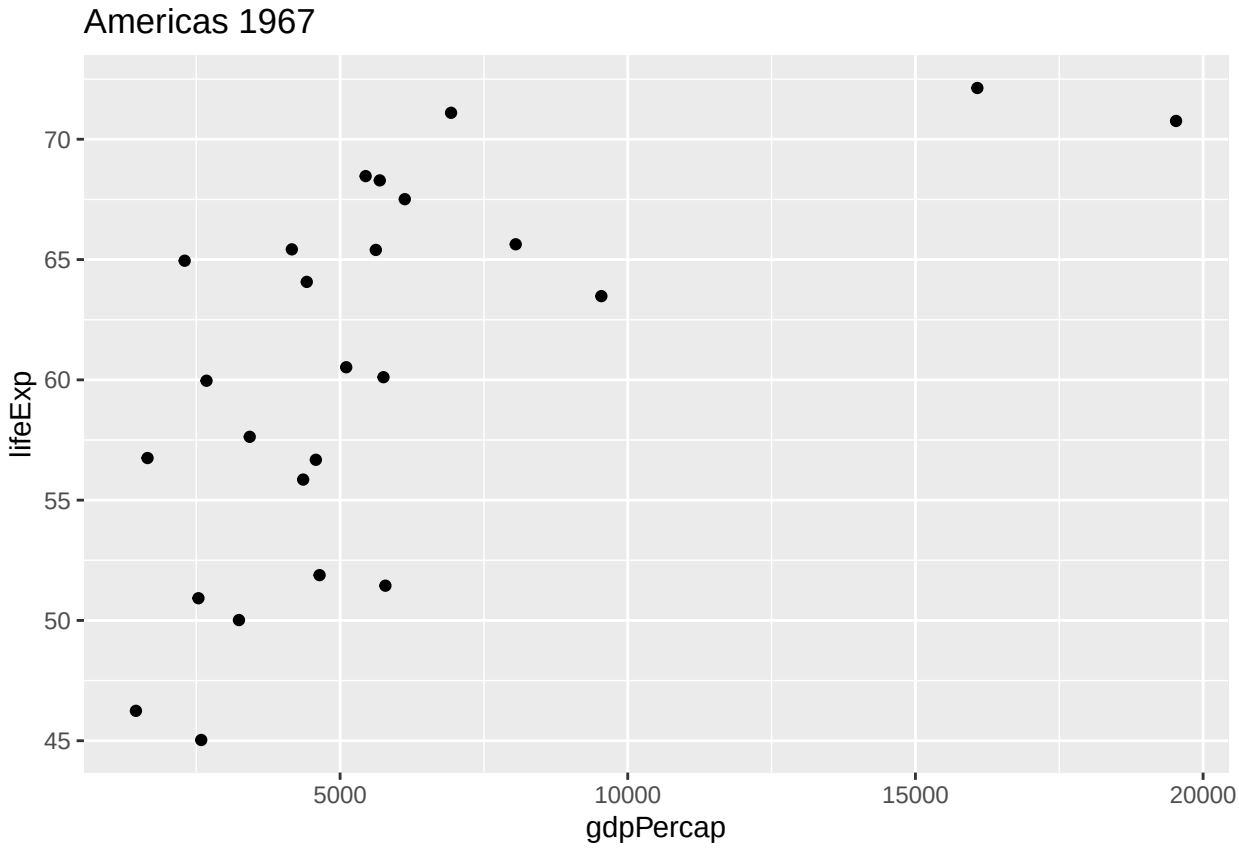

```
##
## [[38]]
```


Americas 1957
70
60
50
40
lifeExp
5000
10000
15000
gdpPercap

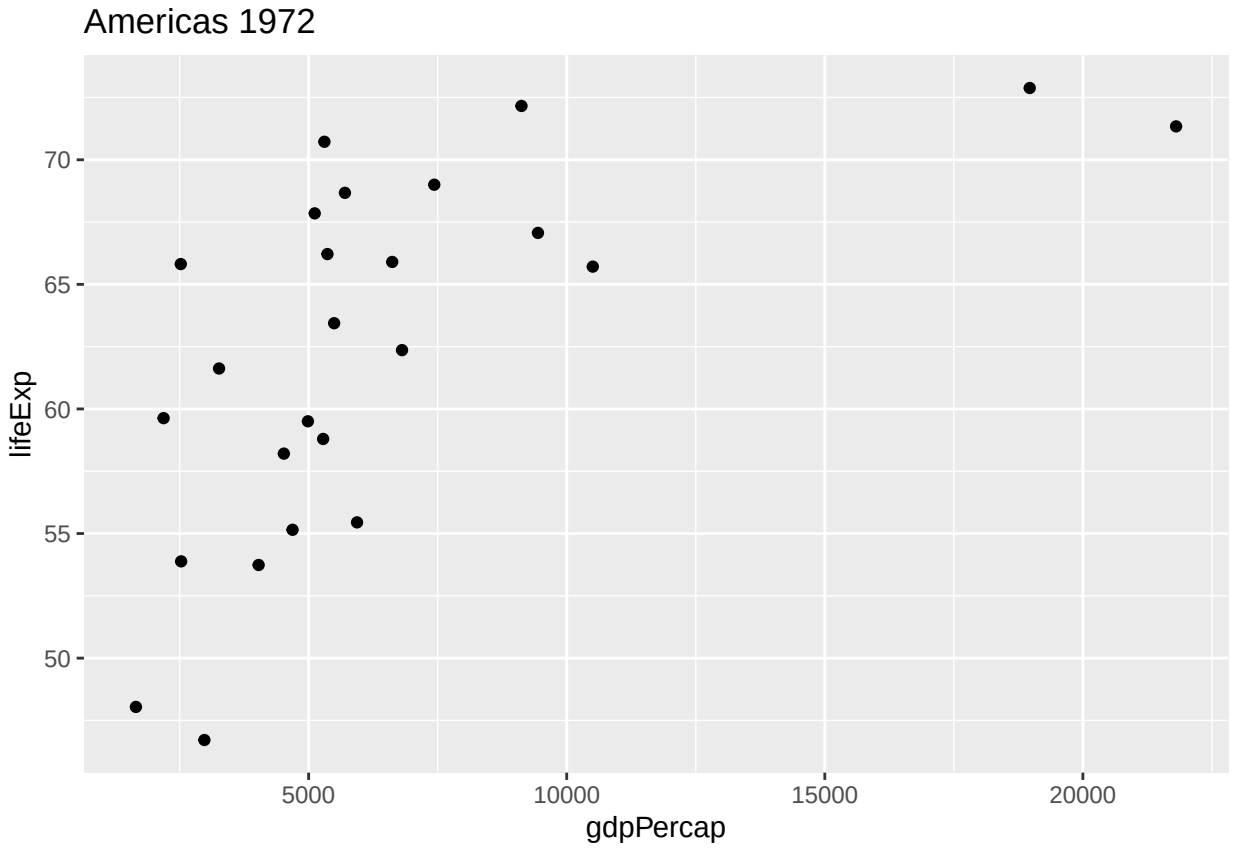
```
##
## [[39]]
```


Americas 1962
70
60
50
lifeExp
4000
8000
12000
16000
gdpPercap

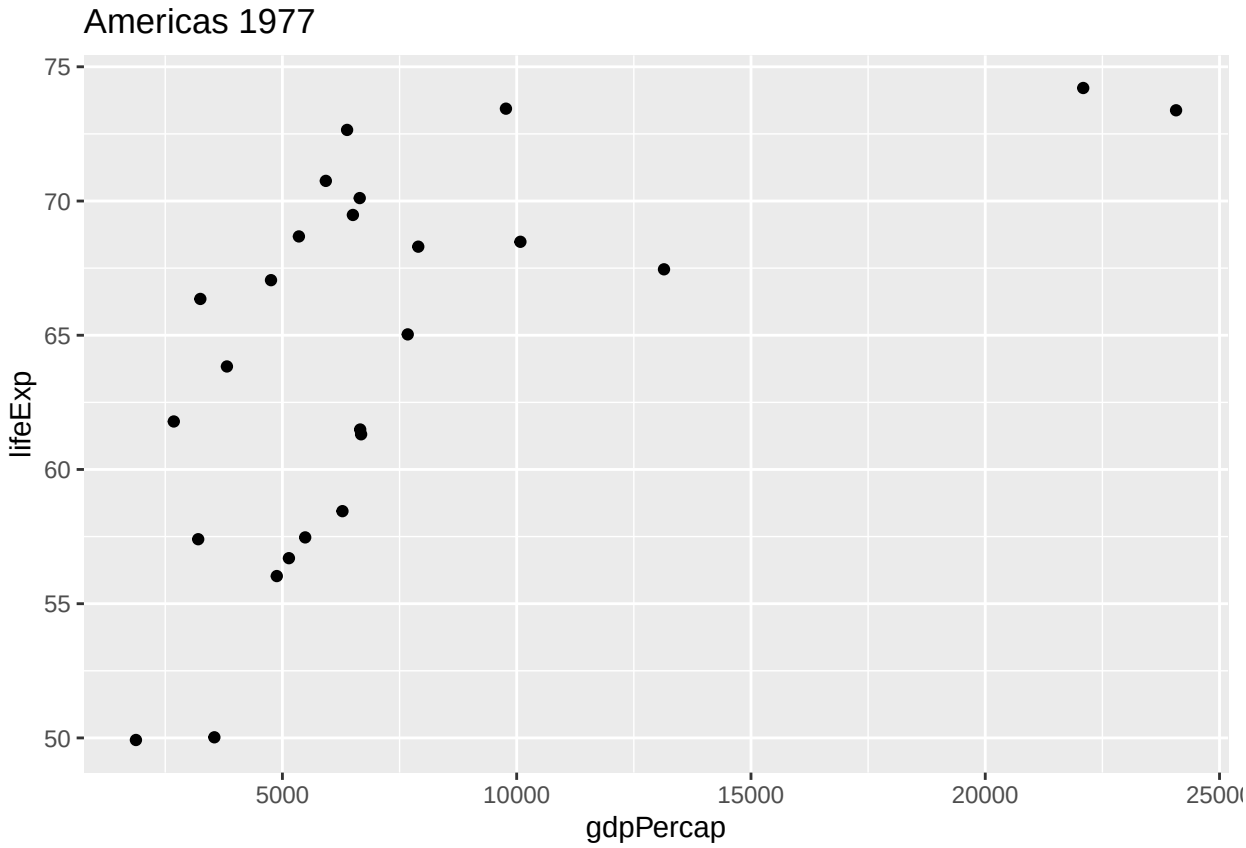
```
##
## [[40]]
```


Americas 1967
lifeExp
gdpPercap
70
65
60
55
50
45
5000
10000
15000
20000

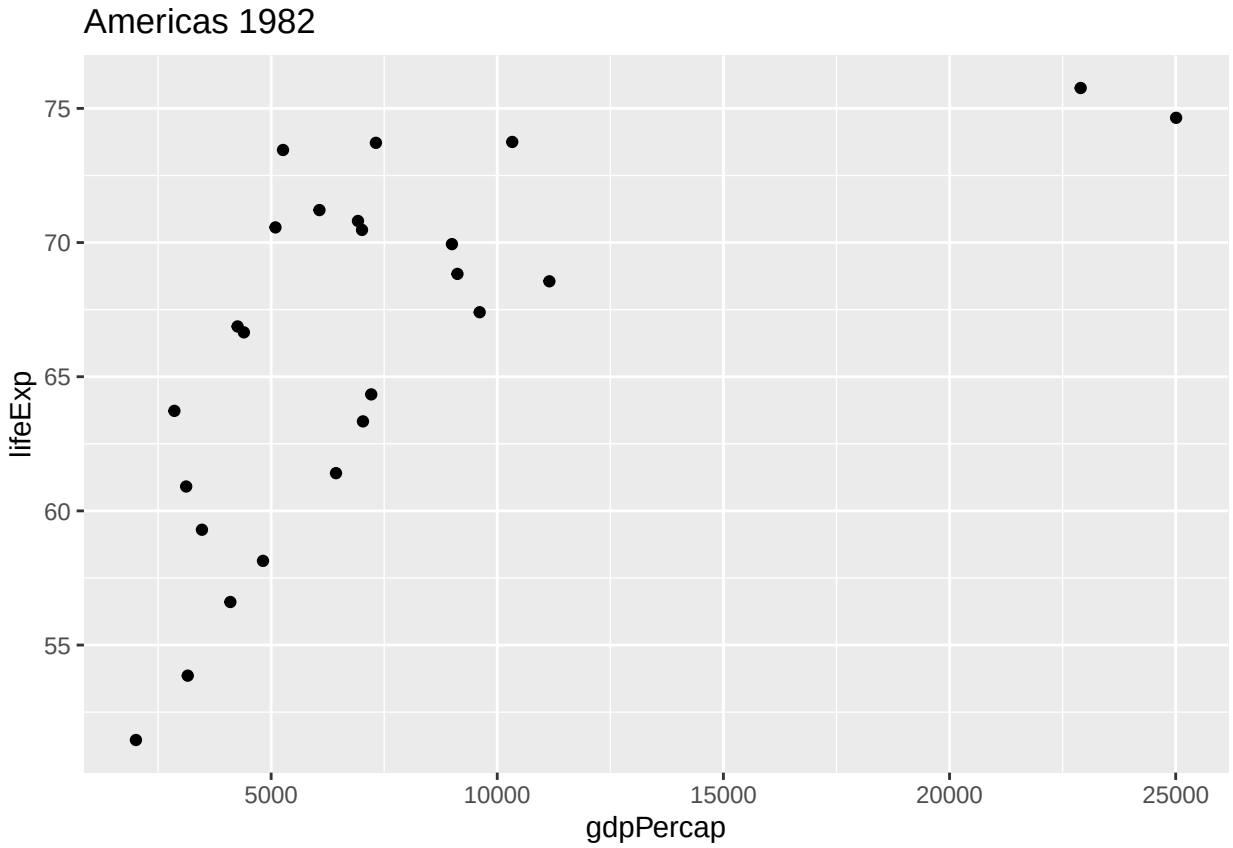
```
##
## [[41]]
```


Americas 1972
lifeExp
70
65
60
55
50
5000
10000
15000
20000
gdpPercap

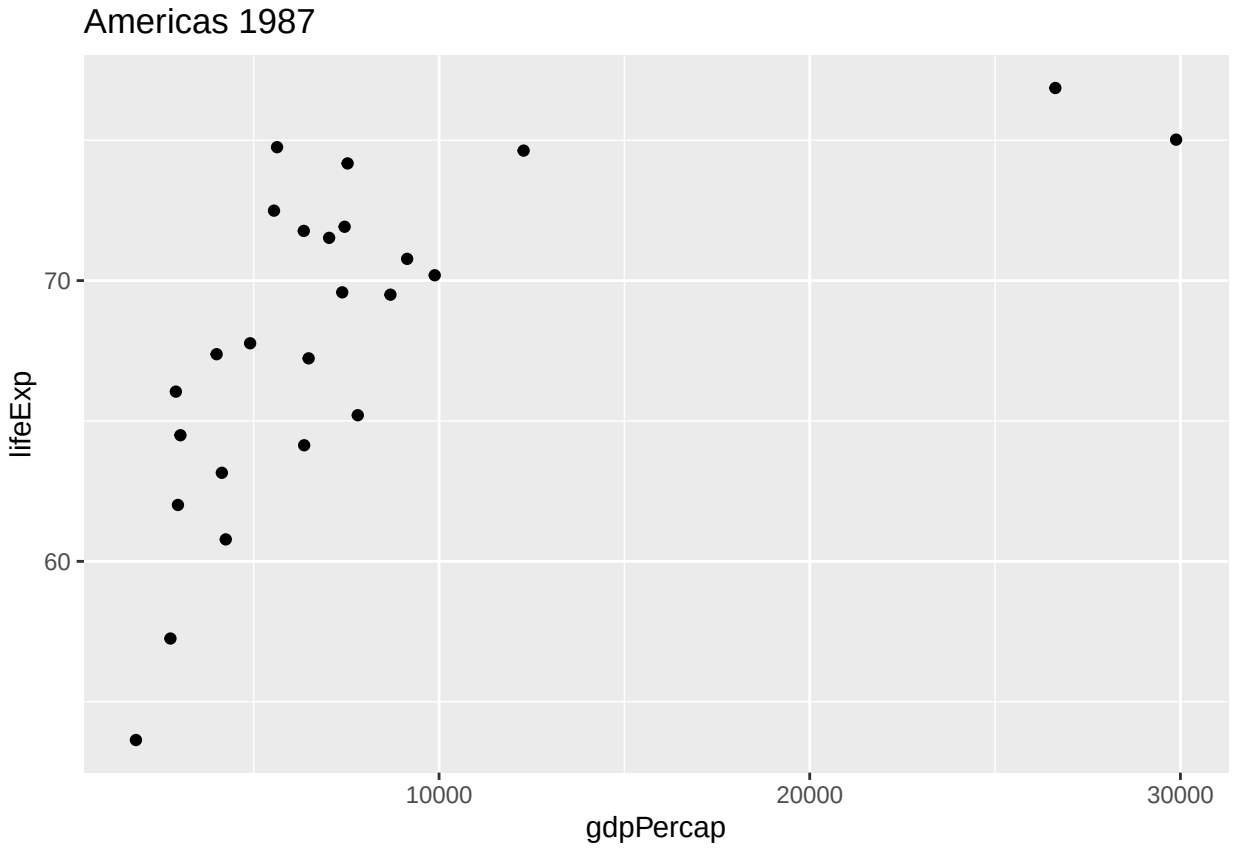
```
##
## [[42]]
```


Americas 1977
75
70
65
60
55
50
lifeExp
5000
10000
15000
20000
gdpPercap

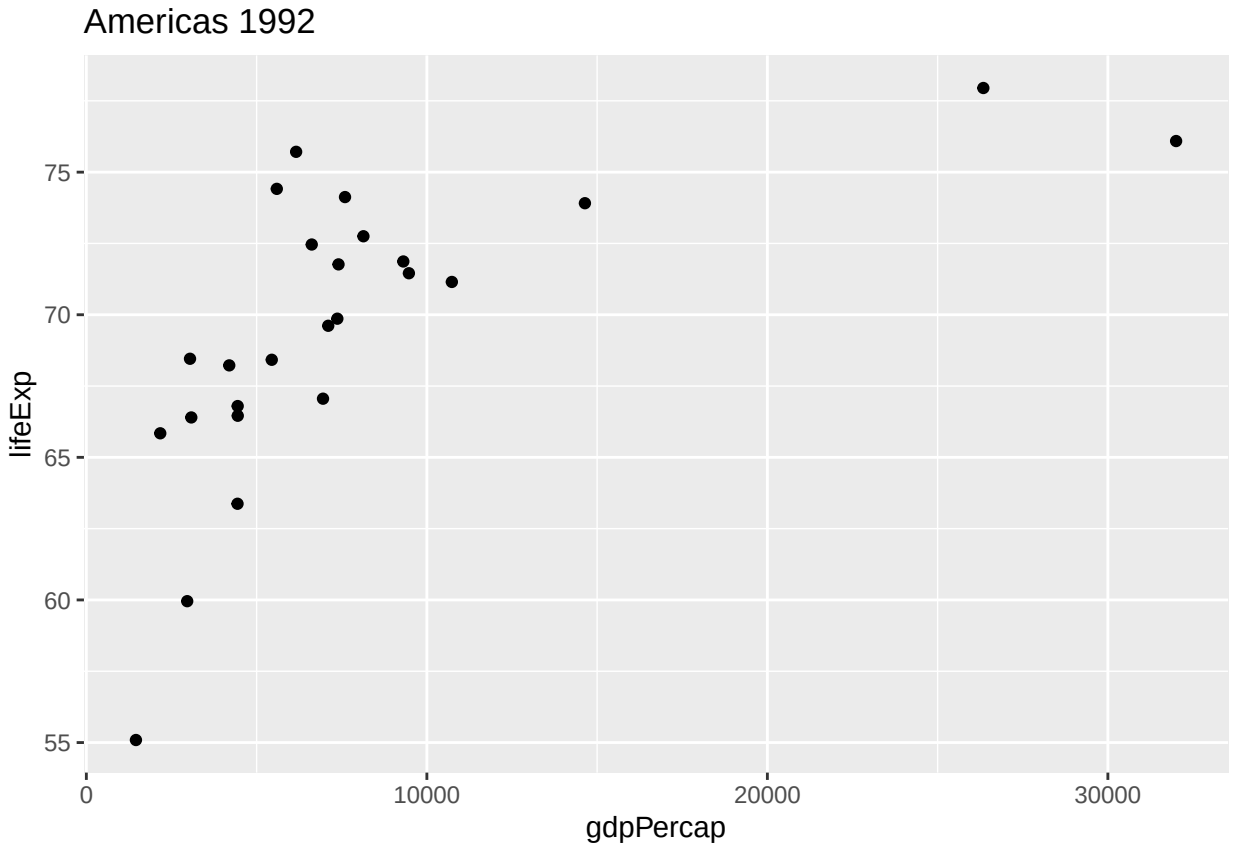
```
##
## [[43]]
```


Americas 1982
lifeExp
75
70
65
60
55
5000
10000
15000
20000
25000
gdpPercap

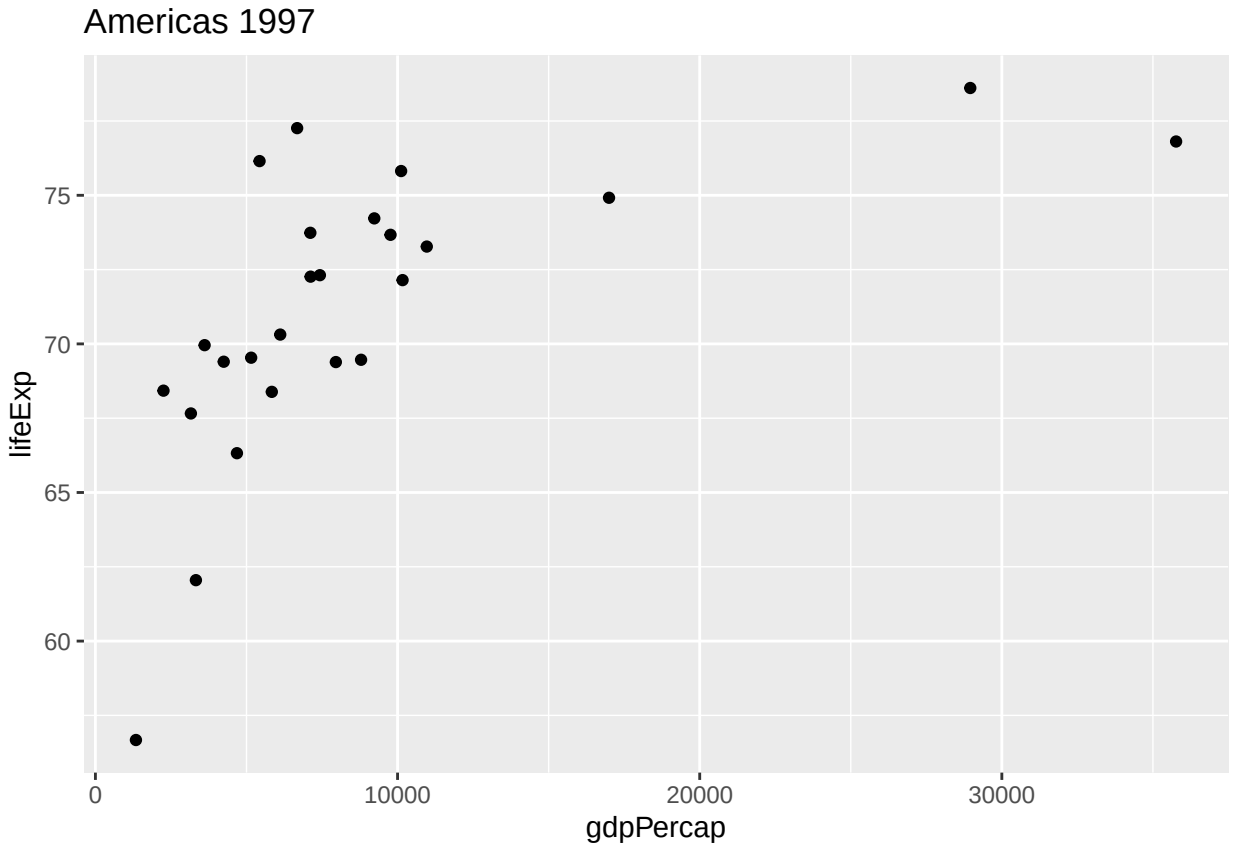
```
##
## [[44]]
```



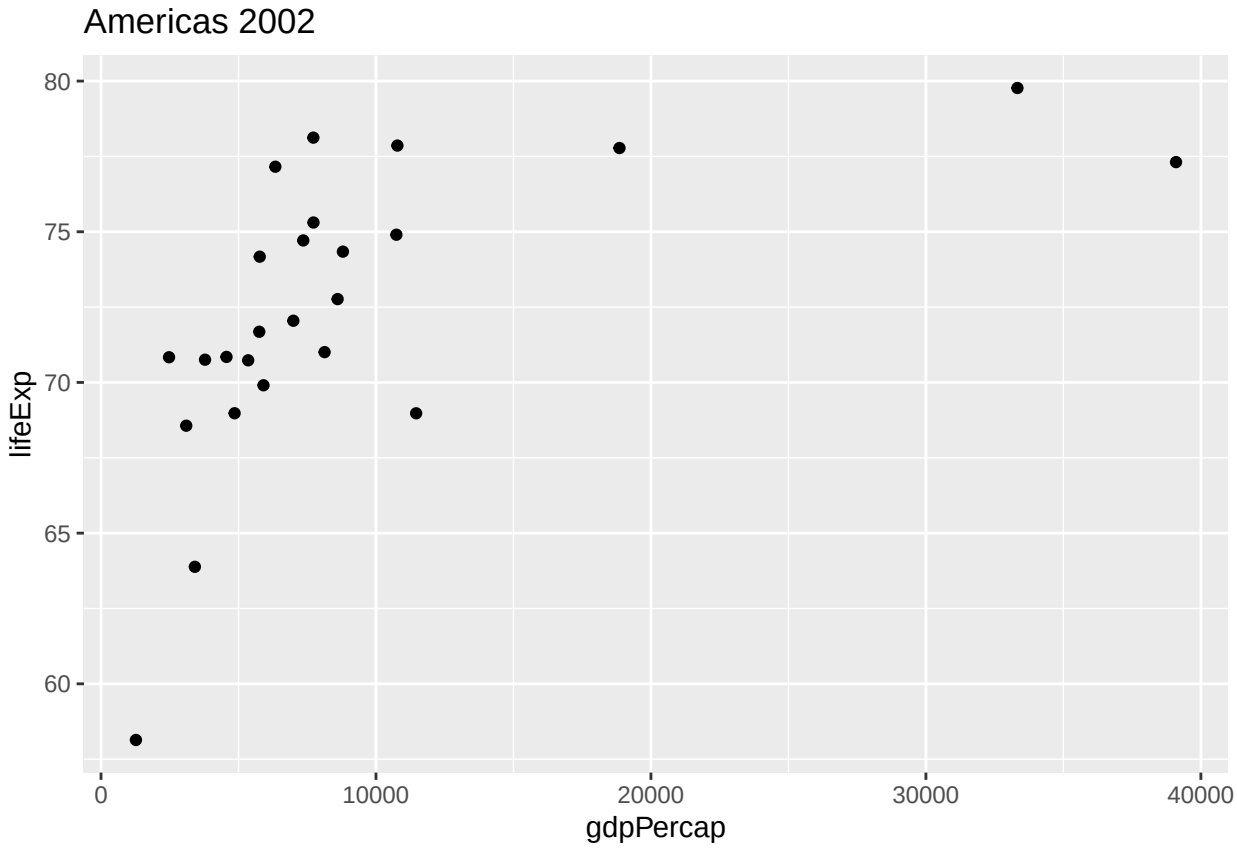

```
##
## [[45]]
```


Americas 1992
lifeExp
gdpPercap
75
70
65
60
55
0
10000
20000
30000

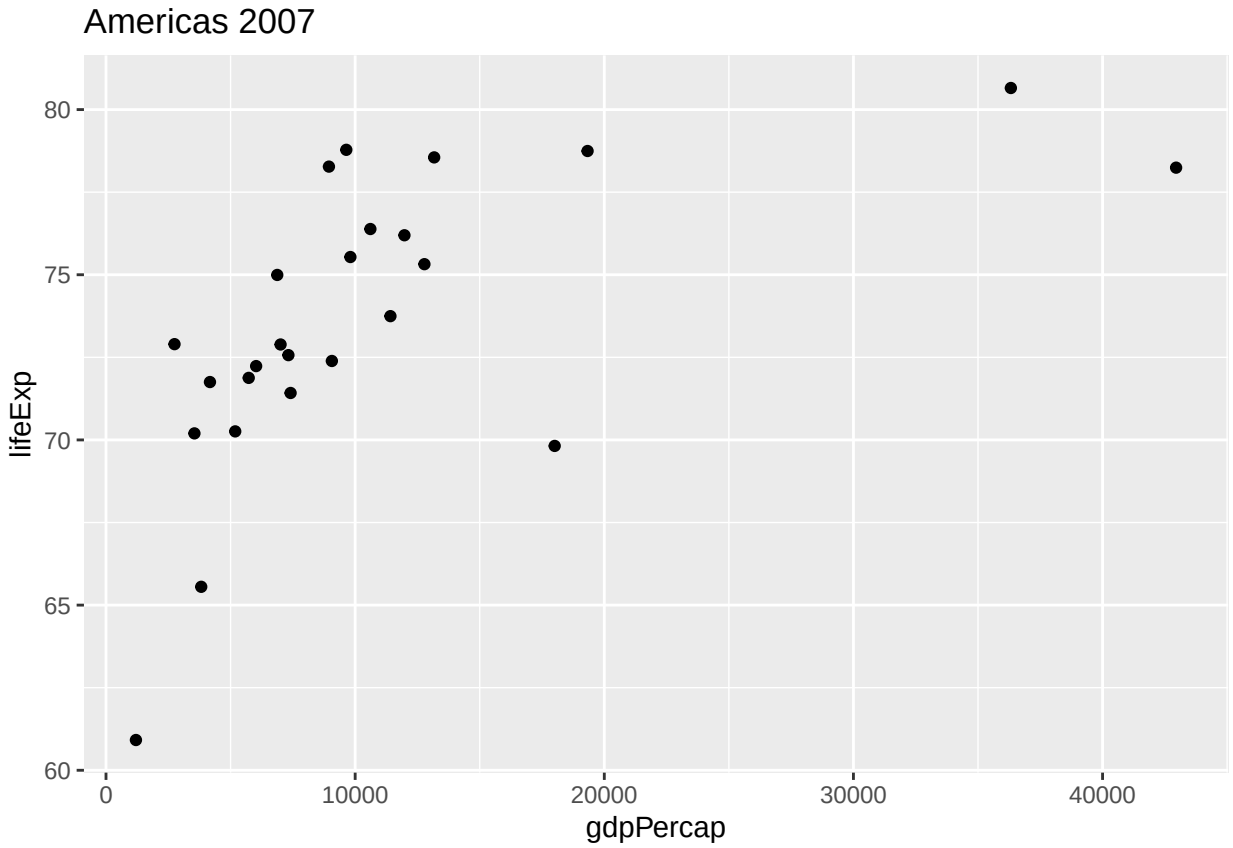
```
##
## [[46]]
```


Americas 1997
lifeExp
75
70
65
60
0
10000
20000
30000
gdpPercap

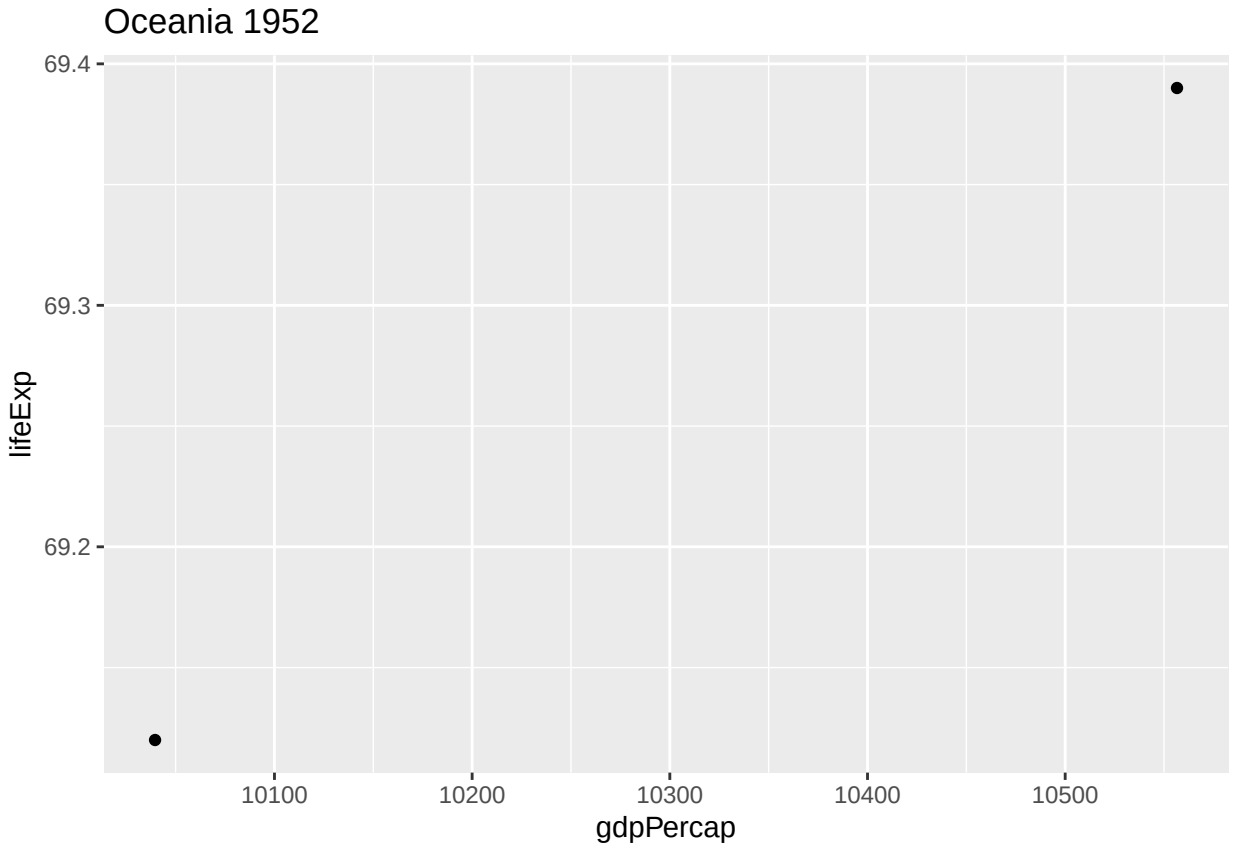
```
##
## [[47]]
```


Americas 2002
lifeExp
gdpPercap
80
75
70
65
60
0
10000
20000
30000
40000

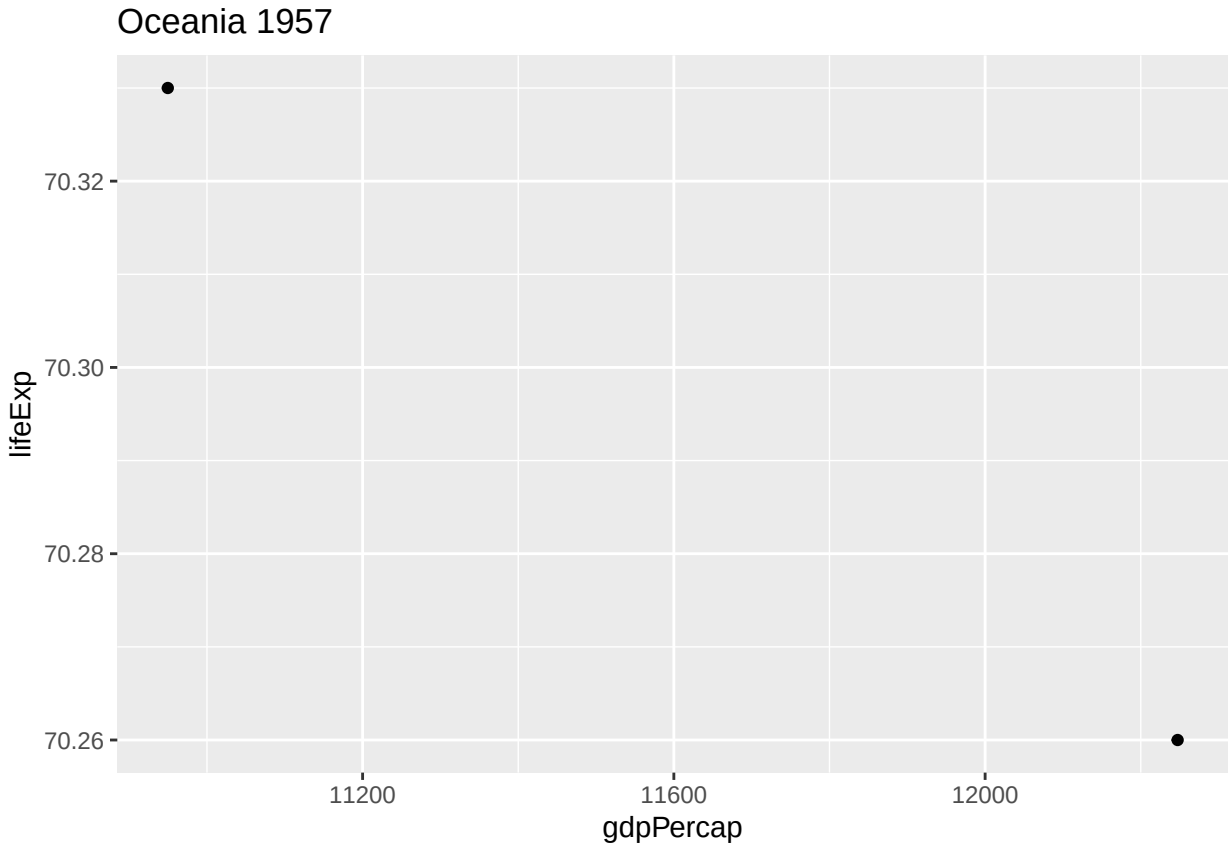
```
##
## [[48]]
```


Americas 2007
lifeExp
gdpPercap
80
75
70
65
60
0
10000
20000
30000
40000

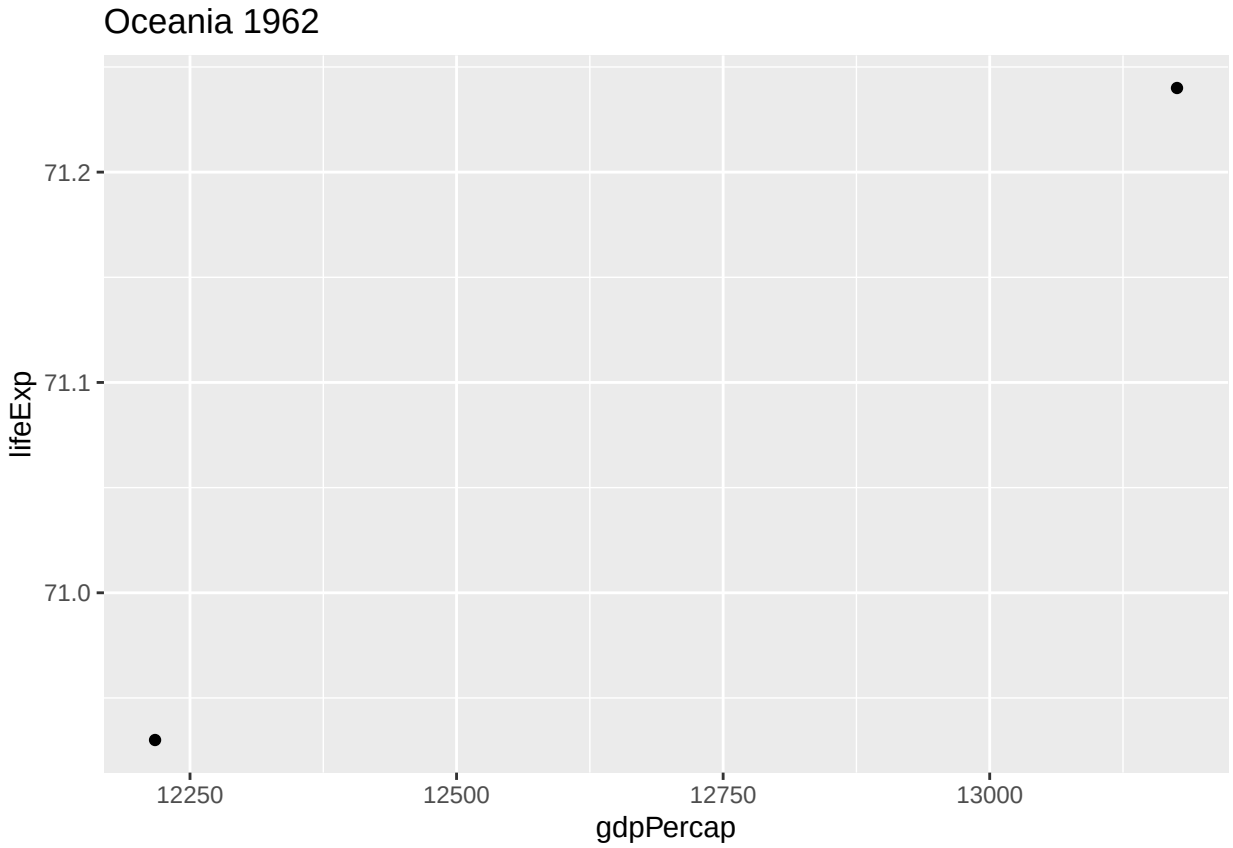
```
##
## [[49]]
```


Oceania 1952
69.4
69.3
69.2
lifeExp
10100
10200
10300
10400
10500
gdpPercap

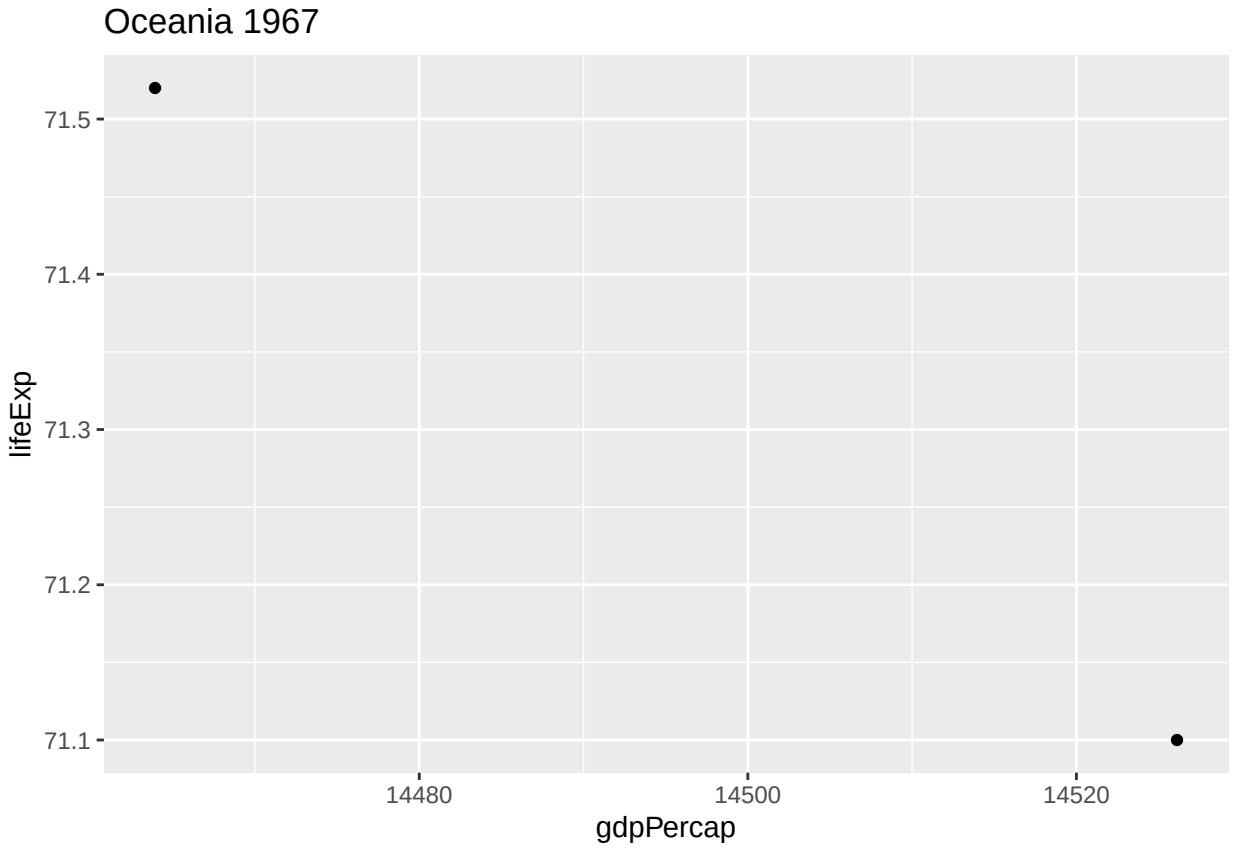
```
##
## [[50]]
```


Oceania 1957
70.32
70.30
70.28
70.26
lifeExp
11200
11600
12000
gdpPercap

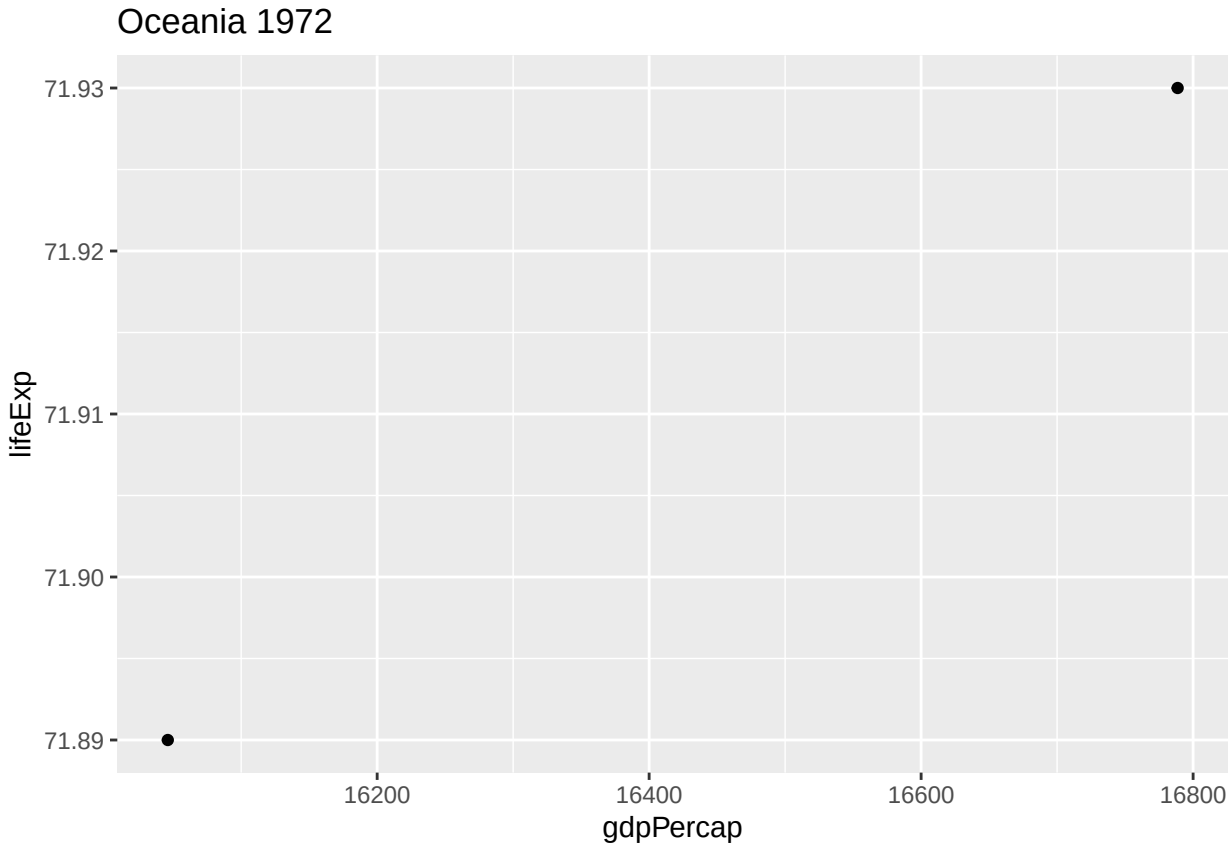
```
##
## [[51]]
```


Oceania 1962
71.2
71.1
71.0
lifeExp
12250
12500
12750
13000
gdpPercap

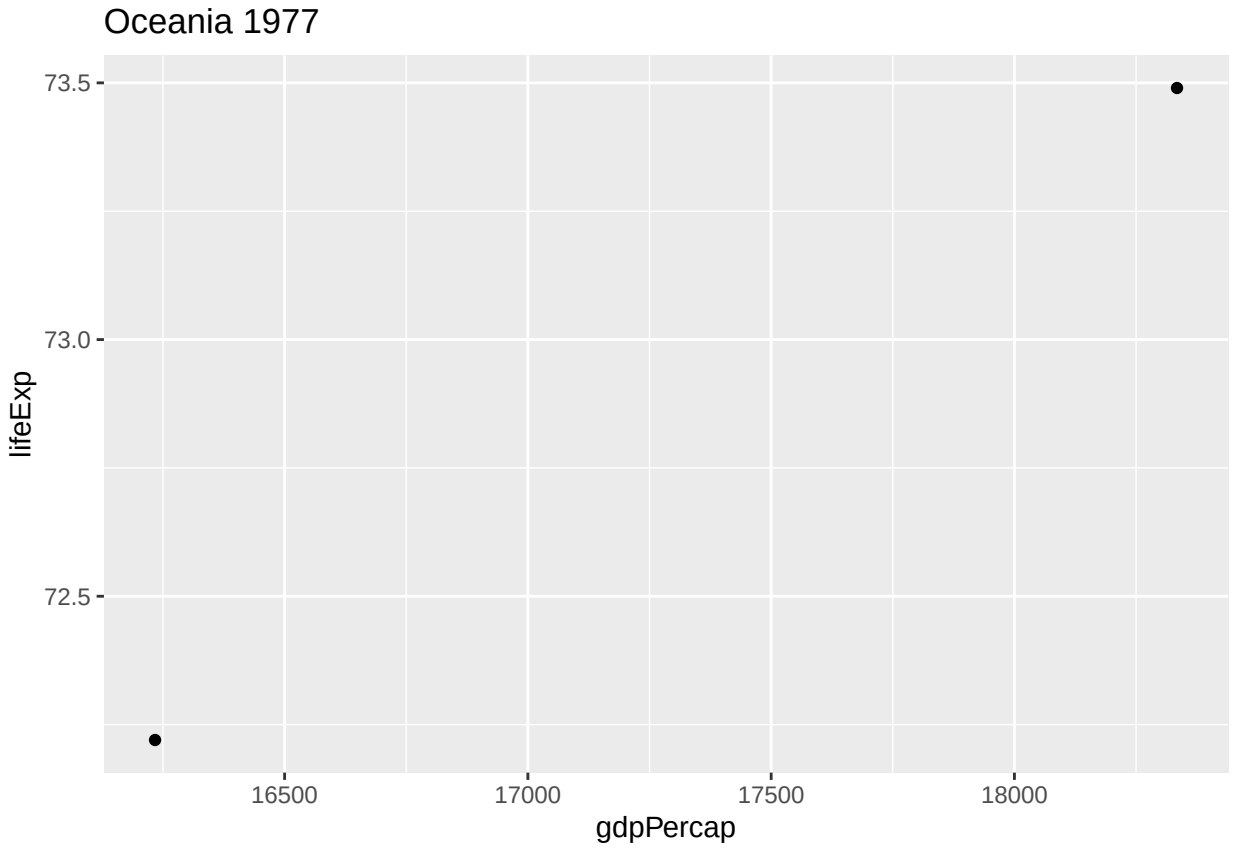
```
##
## [[52]]
```


Oceania 1967
71.5
71.4
71.3
71.2
71.1
lifeExp
14480
14500
14520
gdpPercap

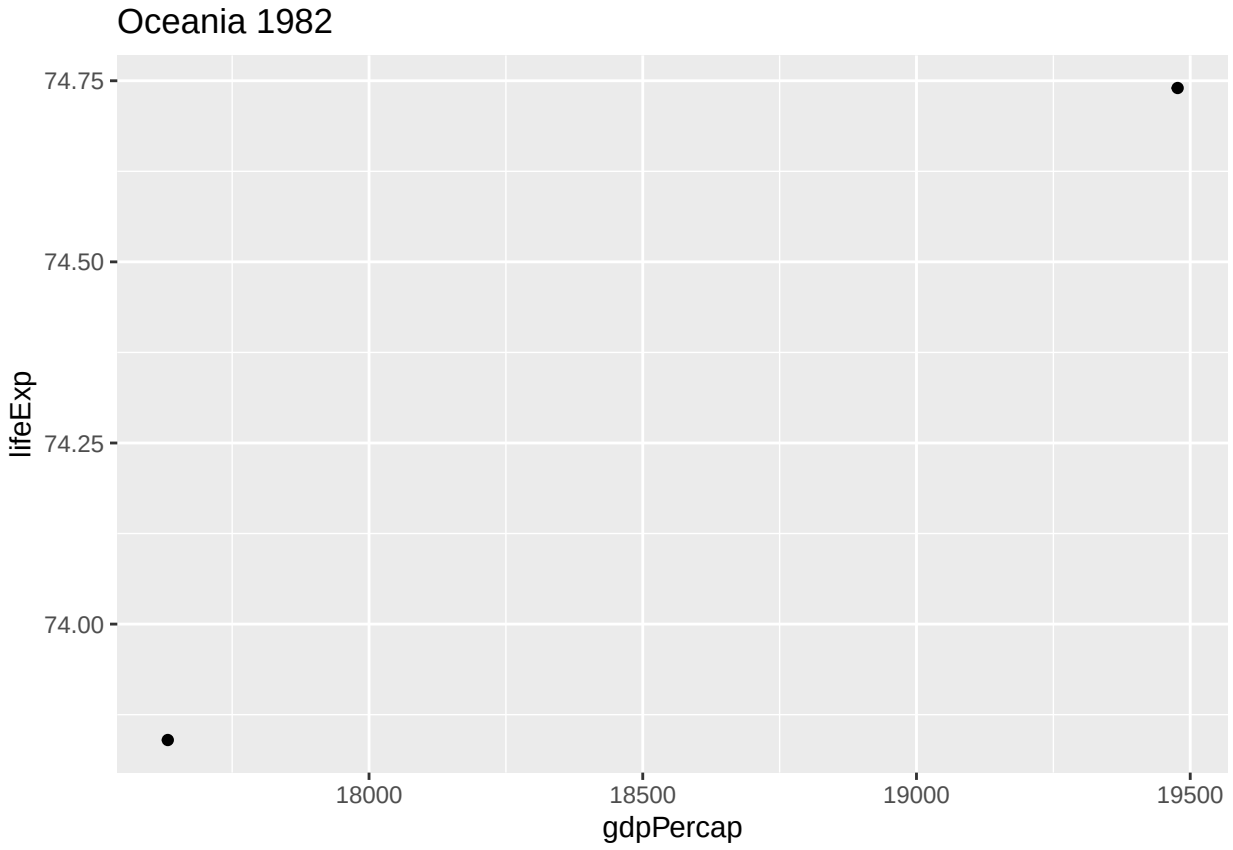
```
##
## [[53]]
```


Oceania 1972
71.93
71.92
71.91
71.90
71.89
lifeExp
16200
16400
16600
16800
gdpPercap

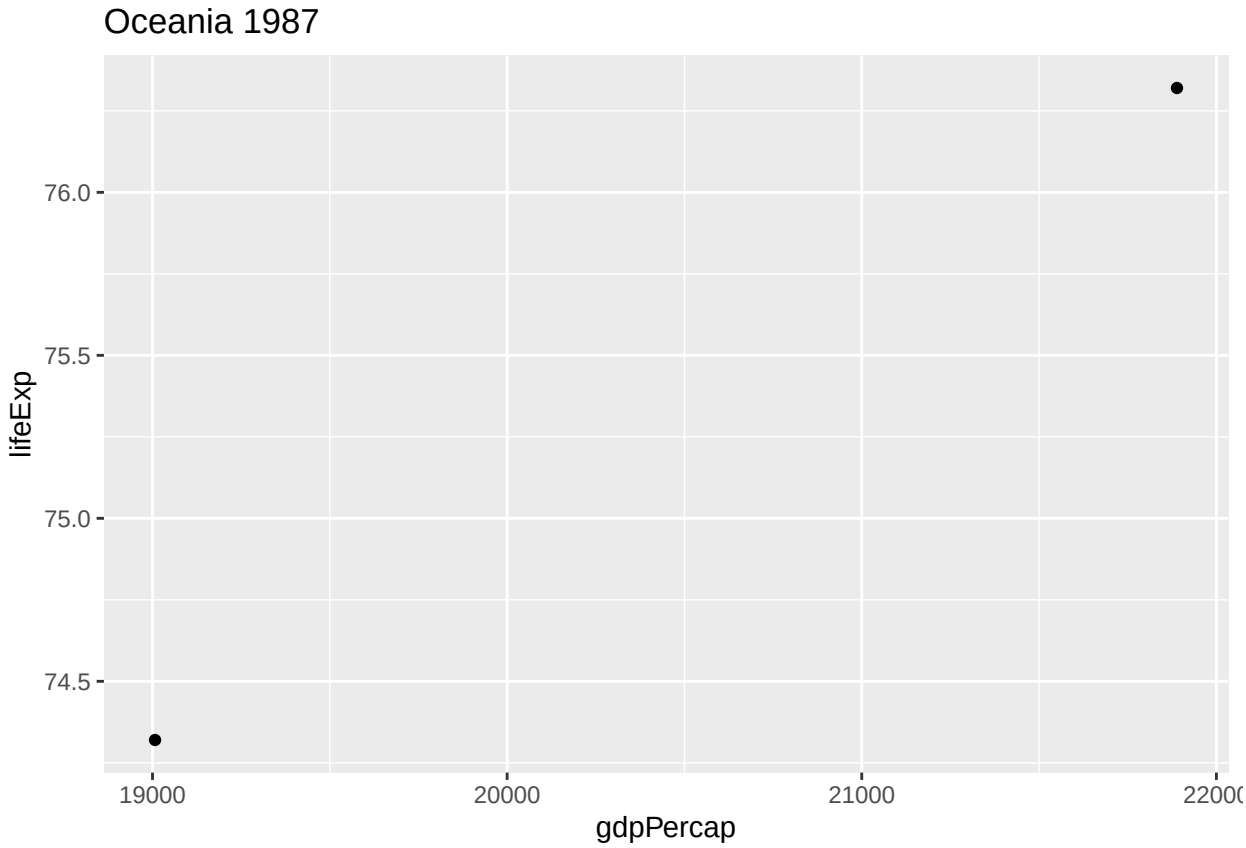
```
##
## [[54]]
```


Oceania 1977
73.5
73.0
72.5
lifeExp
16500
17000
17500
18000
gdpPercap

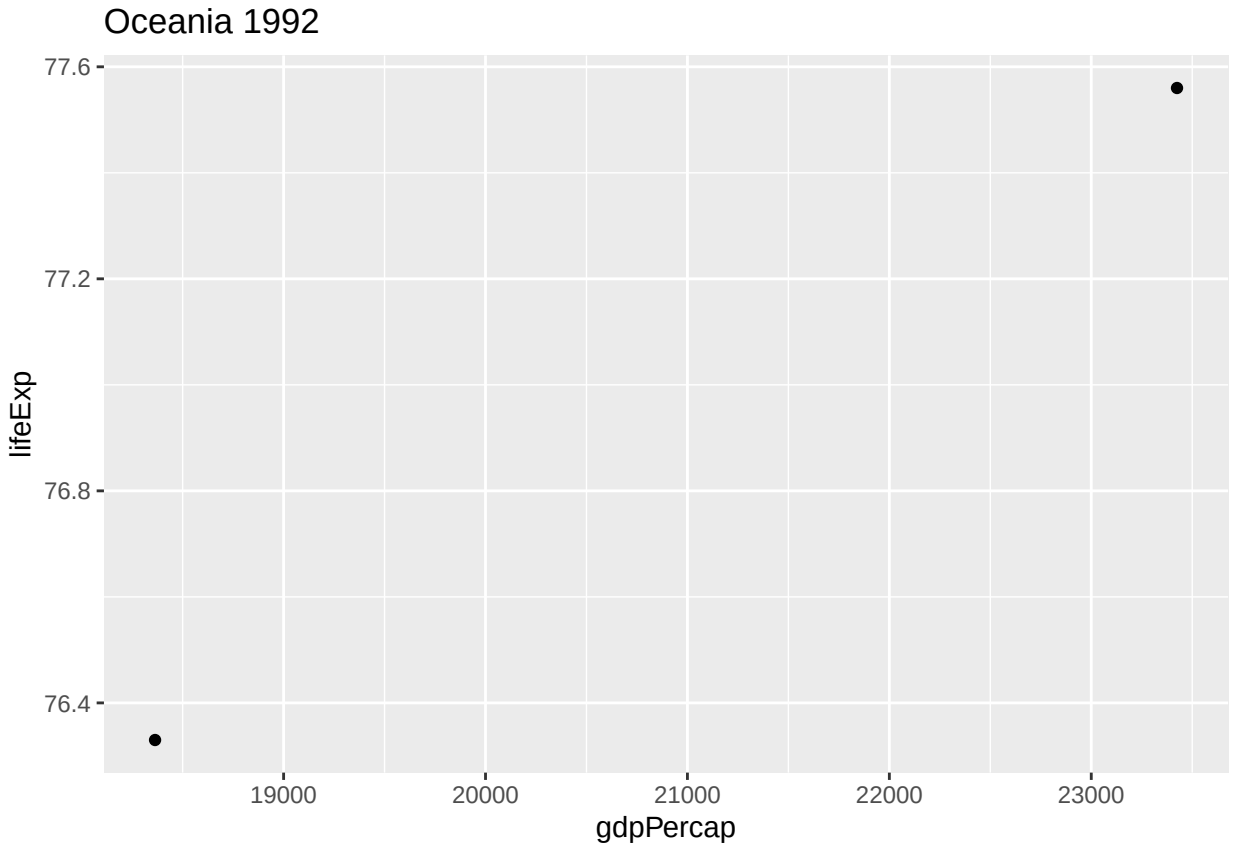
```
##
## [[55]]
```


Oceania 1982
74.75
74.50
74.25
74.00
lifeExp
18000
18500
19000
19500
gdpPercap

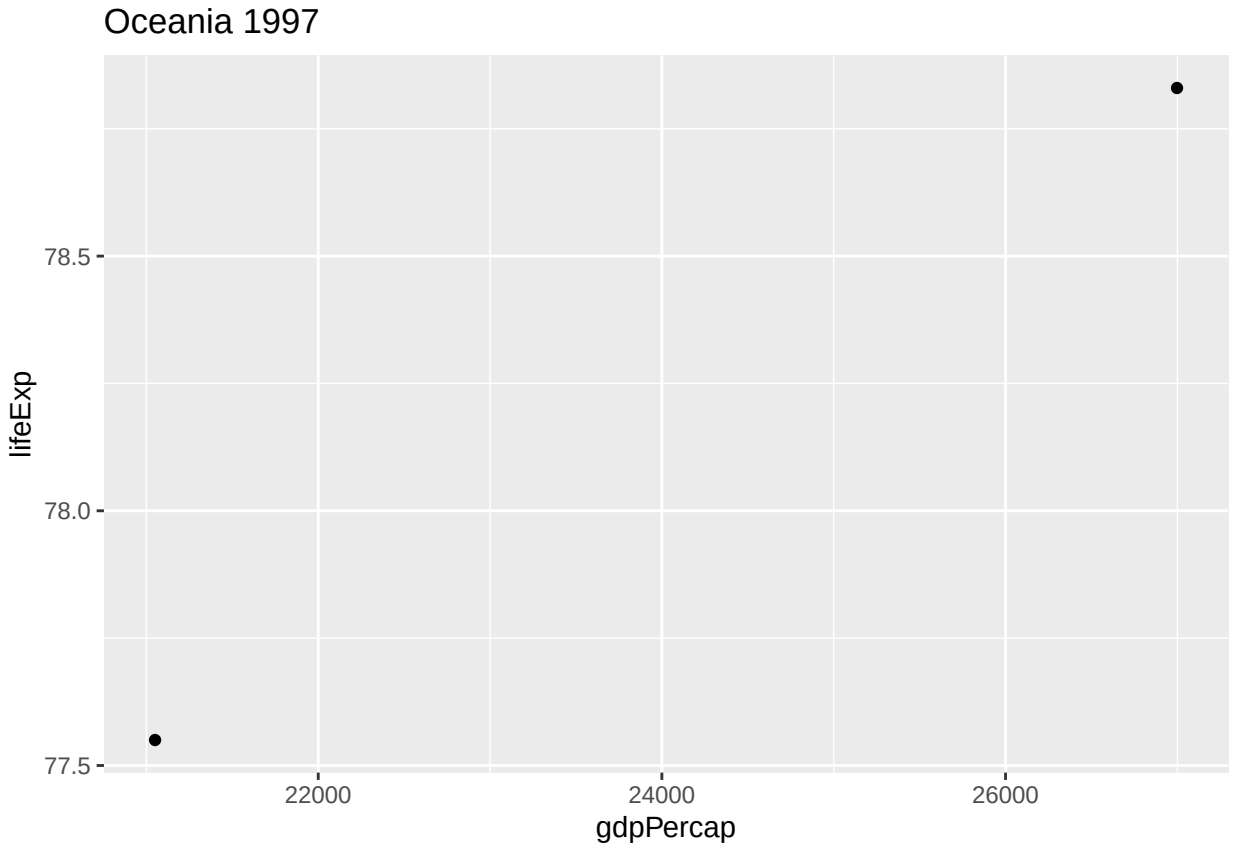
```
##
## [[56]]
```


Oceania 1987
76.0
75.5
75.0
74.5
lifeExp
19000
20000
21000
gdpPercap

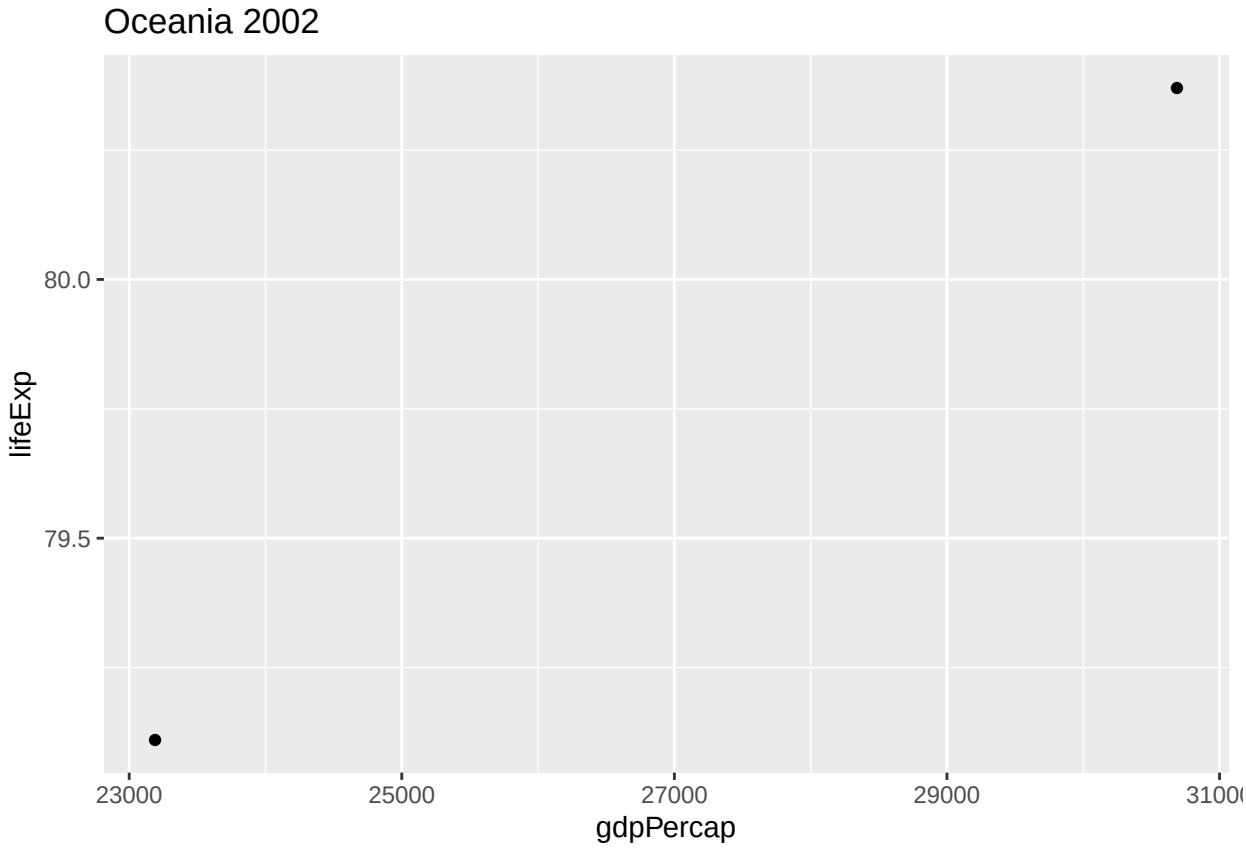
```
##
## [[57]]
```


Oceania 1992
77.6
77.2
76.8
76.4
lifeExp
19000
20000
21000
22000
23000
gdpPercap

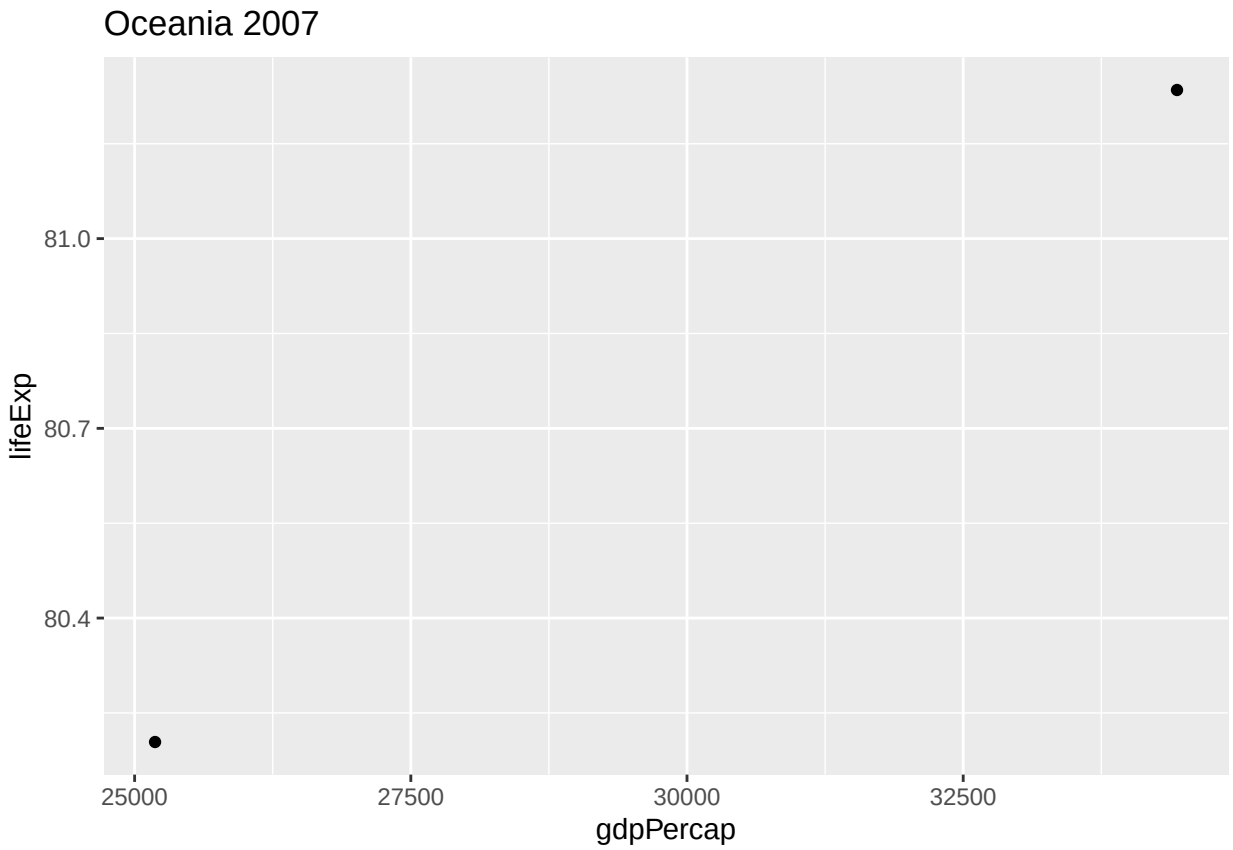
```
##
## [[58]]
```


Oceania 1997
lifeExp
78.5
78.0
77.5
22000
24000
26000
gdpPercap

```
##
## [[59]]
```


Oceania 2002
80.0
79.5
lifeExp
23000
25000
27000
29000
gdpPercap

```
##
## [[60]]
```




```
suppressWarnings(suppressMessages(library(tidyverse)))
library(tidyverse)
is_tibble(gapminder)
```

```
## [1] FALSE
```

```
df_nested <- gapminder %>%
  group_by(continent) %>%
  nest()

is_tibble(df_nested$data[1])
```

```
## [1] FALSE
```

```
str(df_nested$data[1])
```

```
## List of 1
##  $ : tibble [396 x 5] (S3: tbl_df/tbl/data.frame)
##   ..$ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ year     : int [1:396] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
##   ..$ pop      : num [1:396] 8425333 9240934 10267083 11537966 13079460 ...
##   ..$ lifeExp  : num [1:396] 28.8 30.3 32 34 36.1 ...
##   ..$ gdpPercap: num [1:396] 779 821 853 836 740 ...
```

```
# get the nested tibble easily
df_nested %>%
  pluck("data", 5)
```

```
## # A tibble: 24 x 5
##    country    year      pop lifeExp gdpPercap
```

```
##    <fct>     <int>    <dbl>   <dbl>     <dbl>
##  1 Australia  1952  8691212    69.1    10040.
##  2 Australia  1957  9712569    70.3    10950.
##  3 Australia  1962 10794968    70.9    12217.
##  4 Australia  1967 11872264    71.1    14526.
##  5 Australia  1972 13177000    71.9    16789.
##  6 Australia  1977 14074100    73.5    18334.
##  7 Australia  1982 15184200    74.7    19477.
##  8 Australia  1987 16257249    76.3    21889.
##  9 Australia  1992 17481977    77.6    23425.
## 10 Australia  1997 18565243    78.8    26998.
## # ... with 14 more rows
```

```
# fit a separate linear regression model to each continent

df_models <- df_nested %>%
  mutate(lm_model = map(data, ~ lm(lifeExp ~ pop + gdpPercap + year, data=.)))

df_models %>%
  pluck("lm_model", 1)
```

```
##
## Call:
## lm(formula = lifeExp ~ pop + gdpPercap + year, data = .)
##
## Coefficients:
## (Intercept)          pop    gdpPercap         year
##  -7.833e+02    4.228e-11    2.510e-04    4.251e-01
```

```
# now predict on for each tibble (i.e each training data)

df_predict <- df_models %>%
  mutate(predict = map2(lm_model, data, ~ predict(.x, .y)))

# caclulate corresponding MSEs

df_predict %>%
  mutate (mse = map2_dbl(predict, data, ~  mean((.x - .y$lifeExp)^2)))
```

```
## # A tibble: 5 x 5
## # Groups:   continent [5]
##   continent data              lm_model predict        mse
##   <fct>     <list>            <list>   <list>       <dbl>
## 1 Asia      <tibble [396 x 5]> <lm>     <dbl [396]> 67.1
## 2 Europe    <tibble [360 x 5]> <lm>     <dbl [360]>  8.97
## 3 Africa    <tibble [624 x 5]> <lm>     <dbl [624]> 48.8
## 4 Americas  <tibble [300 x 5]> <lm>     <dbl [300]> 34.3
## 5 Oceania   <tibble [24 x 5]>  <lm>     <dbl [24]>   0.368
```

```
#fit a separate linear model for each continent without splitting up the data
helper <- function(df, con){
  model <- lm(lifeExp ~ pop + gdpPercap + year, data=df)
  s <- summary(model)
  print(s)
  cbind(tibble(continent = con),
        as_tibble(s$coefficients, rownames="term")) %>%
```

```
    rename("Std.Error" = "Std. Error", "statistics" = "t value", "pValue" = "Pr(>|t|)")
}

nested <- gapminder %>% group_by(continent) %>% nest()

df_list <- map2(nested$data, nested$continent, ~helper(.x, .y))
```

```
##
## Call:
## lm(formula = lifeExp ~ pop + gdpPercap + year, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -26.1852  -5.2575   0.4857   5.0062  17.9753
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -7.833e+02  4.829e+01 -16.221  < 2e-16 ***
## pop          4.228e-11  2.039e-09   0.021    0.983
## gdpPercap    2.510e-04  3.011e-05   8.336 1.31e-15 ***
## year         4.251e-01  2.442e-02  17.404  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.232 on 392 degrees of freedom
## Multiple R-squared:  0.5222, Adjusted R-squared:  0.5186
## F-statistic: 142.8 on 3 and 392 DF,  p-value: < 2.2e-16
##
##
## Call:
## lm(formula = lifeExp ~ pop + gdpPercap + year, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -21.0315  -0.9954   0.3812   1.6777   5.9680
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.613e+02  2.277e+01  -7.086 7.44e-12 ***
## pop         -8.185e-09  7.798e-09  -1.050    0.295
## gdpPercap    3.255e-04  2.148e-05  15.154  < 2e-16 ***
## year         1.155e-01  1.160e-02   9.961  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.013 on 356 degrees of freedom
## Multiple R-squared:  0.6951, Adjusted R-squared:  0.6926
## F-statistic: 270.6 on 3 and 356 DF,  p-value: < 2.2e-16
##
##
## Call:
## lm(formula = lifeExp ~ pop + gdpPercap + year, data = df)
##
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -26.9053  -4.8255  -0.2523   4.2037  17.5681
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.703e+02  3.386e+01 -13.891   <2e-16 ***
## pop         -3.681e-09  1.888e-08  -0.195    0.845
## gdpPercap    1.121e-03  1.008e-04  11.124   <2e-16 ***
## year         2.610e-01  1.715e-02  15.223   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.009 on 620 degrees of freedom
## Multiple R-squared:  0.4161, Adjusted R-squared:  0.4133
## F-statistic: 147.3 on 3 and 620 DF,  p-value: < 2.2e-16
##
##
## Call:
## lm(formula = lifeExp ~ pop + gdpPercap + year, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.7145  -3.0574   0.7712   3.9654  12.3311
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.333e+02  4.098e+01 -13.013   <2e-16 ***
## pop         -2.151e-08  8.623e-09  -2.494   0.0132 *
## gdpPercap    6.752e-04  7.150e-05   9.444   <2e-16 ***
## year         2.999e-01  2.077e-02  14.441   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.895 on 296 degrees of freedom
## Multiple R-squared:  0.6061, Adjusted R-squared:  0.6021
## F-statistic: 151.8 on 3 and 296 DF,  p-value: < 2.2e-16
##
##
## Call:
## lm(formula = lifeExp ~ pop + gdpPercap + year, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.2205 -0.4987  0.2728  0.4372  0.7368
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.097e+02  5.119e+01  -4.097 0.000561 ***
## pop          8.365e-09  3.335e-08   0.251 0.804526
## gdpPercap    2.027e-04  8.466e-05   2.395 0.026554 *
## year         1.415e-01  2.652e-02   5.337 3.19e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.6648 on 20 degrees of freedom
## Multiple R-squared:  0.9733, Adjusted R-squared:  0.9693
## F-statistic: 243.3 on 3 and 20 DF,  p-value: 6.663e-16
```

```
df_list %>% reduce(rbind)
```

```
##    continent        term      Estimate    Std.Error   statistics       pValue
## 1       Asia (Intercept) -7.833346e+02 4.829159e+01 -16.22093133 1.221345e-45
## 2       Asia         pop  4.228480e-11 2.038675e-09   0.02074131 9.834626e-01
## 3       Asia   gdpPercap  2.510235e-04 3.011313e-05   8.33601577 1.313850e-15
## 4       Asia        year  4.250632e-01 2.442390e-02  17.40357584 1.130117e-50
## 5     Europe (Intercept) -1.613479e+02 2.276954e+01  -7.08613154 7.439346e-12
## 6     Europe         pop -8.184650e-09 7.797942e-09  -1.04959092 2.946182e-01
## 7     Europe   gdpPercap  3.254890e-04 2.147815e-05  15.15442184 2.213007e-40
## 8     Europe        year  1.155254e-01 1.159828e-02   9.96055931 8.880118e-21
## 9     Africa (Intercept) -4.702809e+02 3.385505e+01 -13.89101111 2.165718e-38
## 10    Africa         pop -3.681098e-09 1.887800e-08  -0.19499402 8.454615e-01
## 11    Africa   gdpPercap  1.121480e-03 1.008143e-04  11.12421382 2.460256e-26
## 12    Africa        year  2.610369e-01 1.714719e-02  15.22330342 1.074461e-44
## 13  Americas (Intercept) -5.333126e+02 4.098462e+01 -13.01250536 6.399123e-31
## 14  Americas         pop -2.150868e-08 8.623439e-09  -2.49421169 1.316914e-02
## 15  Americas   gdpPercap  6.752172e-04 7.149908e-05   9.44371867 1.125154e-18
## 16  Americas        year  2.999141e-01 2.076844e-02  14.44085980 3.789314e-36
## 17   Oceania (Intercept) -2.097021e+02 5.118993e+01  -4.09655043 5.613344e-04
## 18   Oceania         pop  8.365083e-09 3.335332e-08   0.25080209 8.045257e-01
## 19   Oceania   gdpPercap  2.027306e-04 8.466481e-05   2.39450851 2.655396e-02
## 20   Oceania        year  1.415402e-01 2.652184e-02   5.33674265 3.185148e-05
```

```
# a better solution
gapminder %>%
  group_by(continent) %>%
  nest() %>%
  mutate(lm_obj = map(data, ~lm(lifeExp ~ pop + year + gdpPercap, data = .))) %>%
  mutate(lm_tidy = map(lm_obj, broom::tidy)) %>%
  ungroup() %>% # on next line we use "continent" column that we grouped on
  transmute(continent, lm_tidy) %>%
  unnest(cols = c(lm_tidy)) # explode nested tibbles rows into containing df
```

```
## # A tibble: 20 x 6
##    continent term         estimate std.error statistic  p.value
##    <fct>     <chr>           <dbl>     <dbl>     <dbl>    <dbl>
##  1 Asia      (Intercept) -7.83e+ 2   4.83e+1  -16.2    1.22e-45
##  2 Asia      pop          4.23e-11   2.04e-9    0.0207 9.83e- 1
##  3 Asia      year         4.25e- 1   2.44e-2   17.4    1.13e-50
##  4 Asia      gdpPercap    2.51e- 4   3.01e-5    8.34   1.31e-15
##  5 Europe    (Intercept) -1.61e+ 2   2.28e+1   -7.09   7.44e-12
##  6 Europe    pop         -8.18e- 9   7.80e-9   -1.05   2.95e- 1
##  7 Europe    year         1.16e- 1   1.16e-2    9.96   8.88e-21
##  8 Europe    gdpPercap    3.25e- 4   2.15e-5   15.2    2.21e-40
##  9 Africa    (Intercept) -4.70e+ 2   3.39e+1  -13.9    2.17e-38
## 10 Africa    pop         -3.68e- 9   1.89e-8   -0.195  8.45e- 1
## 11 Africa    year         2.61e- 1   1.71e-2   15.2    1.07e-44
## 12 Africa    gdpPercap    1.12e- 3   1.01e-4   11.1    2.46e-26
## 13 Americas  (Intercept) -5.33e+ 2   4.10e+1  -13.0    6.40e-31
## 14 Americas  pop         -2.15e- 8   8.62e-9   -2.49   1.32e- 2
```

```
## 15 Americas  year         3.00e- 1   2.08e-2   14.4    3.79e-36
## 16 Americas  gdpPercap    6.75e- 4   7.15e-5    9.44   1.13e-18
## 17 Oceania   (Intercept) -2.10e+ 2   5.12e+1   -4.10   5.61e- 4
## 18 Oceania   pop          8.37e- 9   3.34e-8    0.251  8.05e- 1
## 19 Oceania   year         1.42e- 1   2.65e-2    5.34   3.19e- 5
## 20 Oceania   gdpPercap    2.03e- 4   8.47e-5    2.39   2.66e- 2
```

```
# To apply mutate functions to a list-column (i.e. a tibble column)
# you need to wrap the function you want to apply in a map function.

df_nested %>%
  mutate(dimensions = glue::glue(map_int(data,nrow), " X ", map_int(data,ncol)))
```

```
## Warning in mutate_impl(.data, dots, caller_env()): Vectorizing 'glue' elements
## may not preserve their attributes

## Warning in mutate_impl(.data, dots, caller_env()): Vectorizing 'glue' elements
## may not preserve their attributes

## Warning in mutate_impl(.data, dots, caller_env()): Vectorizing 'glue' elements
## may not preserve their attributes

## Warning in mutate_impl(.data, dots, caller_env()): Vectorizing 'glue' elements
## may not preserve their attributes

## Warning in mutate_impl(.data, dots, caller_env()): Vectorizing 'glue' elements
## may not preserve their attributes
```

```
## # A tibble: 5 x 3
## # Groups:   continent [5]
##   continent data              dimensions
##   <fct>     <list>            <chr>
## 1 Asia      <tibble [396 x 5]> 396 X 5
## 2 Europe    <tibble [360 x 5]> 360 X 5
## 3 Africa    <tibble [624 x 5]> 624 X 5
## 4 Americas  <tibble [300 x 5]> 300 X 5
## 5 Oceania   <tibble [24 x 5]>  24 X 5
```

```
# another example, calculate average life expentency in each continit
df_nested %>%
  mutate(life_expect = map_dbl(data, ~ mean(.$lifeExp)))
```

```
## # A tibble: 5 x 3
## # Groups:   continent [5]
##   continent data              life_expect
##   <fct>     <list>                  <dbl>
## 1 Asia      <tibble [396 x 5]>       60.1
## 2 Europe    <tibble [360 x 5]>       71.9
## 3 Africa    <tibble [624 x 5]>       48.9
## 4 Americas  <tibble [300 x 5]>       64.7
## 5 Oceania   <tibble [24 x 5]>        74.3
```

```
suppressWarnings(suppressMessages(library(tidyverse)))
library(tidyverse)
# map(element's index) extracts an element in the index from all nested lists
```

```r
list(ids = 1:3, values=c("one", "two", "three")) %>%
  map(2)
```

```
## $ids
## [1] 2
##
## $values
## [1] "two"
```

```r
# read list of csv files with the same schema and merge them
list.files("../open-data/", pattern = "^2017", full.names = TRUE) %>%
  map_df(read_csv)
```

```
## # A tibble: 0 x 0
```

```r
# split simply group-by on given column and place them in separate tibbles
gapminder_list <- gapminder %>%
  split(gapminder$continent)


# Sample 5 records from each continent tibble
samples <- gapminder %>%
  split(gapminder$continent) %>%
  map(~(sample_n(., 5)))
samples
```

```
## $Africa
##   country year      pop continent lifeExp gdpPercap
## 1  Uganda 1967  8900294    Africa  48.051  908.9185
## 2   Niger 1987  7332638    Africa  44.555  668.3000
## 3   Egypt 1997 66134291    Africa  67.217 4173.1818
## 4 Nigeria 1992 93364244    Africa  47.472 1619.8482
## 5 Burundi 2007  8390505    Africa  49.580  430.0707
##
## $Americas
##         country year       pop continent lifeExp gdpPercap
## 1       Jamaica 2007   2780132  Americas  72.567  7320.880
## 2 United States 2007 301139947  Americas  78.242 42951.653
## 3     Nicaragua 1987   3344353  Americas  62.008  2955.984
## 4       Jamaica 1957   1535090  Americas  62.610  4756.526
## 5     Guatemala 1967   4690773  Americas  50.016  3242.531
##
## $Asia
##              country year     pop continent lifeExp  gdpPercap
## 1           Cambodia 1982 7272485      Asia  50.957   624.4755
## 2 West Bank and Gaza 1987 1691210      Asia  67.046  5107.1974
## 3            Lebanon 2007 3921278      Asia  71.993 10461.0587
## 4    Hong Kong China 1982 5264500      Asia  75.450 14560.5305
## 5           Mongolia 1962 1010280      Asia  48.251  1056.3540
##
## $Europe
##   country year      pop continent lifeExp gdpPercap
## 1 Austria 1967  7376998    Europe  70.140 12834.602
## 2 Romania 1972 20662648    Europe  69.210  8011.414
## 3  Turkey 2007 71158647    Europe  71.777  8458.276
```

```
## 4 Albania 2007  3600523    Europe  76.423  5937.030
## 5  Norway 1957  3491938    Europe  73.440 11653.973
##
## $Oceania
##       country year      pop continent lifeExp gdpPercap
## 1 New Zealand 1962  2488550   Oceania   71.24  13175.68
## 2   Australia 1977 14074100   Oceania   73.49  18334.20
## 3 New Zealand 1967  2728150   Oceania   71.52  14463.92
## 4 New Zealand 1952  1994794   Oceania   69.39  10556.58
## 5 New Zealand 1992  3437674   Oceania   76.33  18363.32
```

```
# keep those list elements (here datasets) that satisfy certain coditions
samples %>%
  keep(~(mean(.$lifeExp) > 70))
```

```
## $Europe
##   country year      pop continent lifeExp gdpPercap
## 1 Austria 1967  7376998    Europe  70.140 12834.602
## 2 Romania 1972 20662648    Europe  69.210  8011.414
## 3  Turkey 2007 71158647    Europe  71.777  8458.276
## 4 Albania 2007  3600523    Europe  76.423  5937.030
## 5  Norway 1957  3491938    Europe  73.440 11653.973
##
## $Oceania
##       country year      pop continent lifeExp gdpPercap
## 1 New Zealand 1962  2488550   Oceania   71.24  13175.68
## 2   Australia 1977 14074100   Oceania   73.49  18334.20
## 3 New Zealand 1967  2728150   Oceania   71.52  14463.92
## 4 New Zealand 1952  1994794   Oceania   69.39  10556.58
## 5 New Zealand 1992  3437674   Oceania   76.33  18363.32
```

```
# discard with some magical stringr
"absc61, sf22ve23,NA, wefc,wfverv,3rf3f344,rffr3$f446,,,rf11,NA,345fv,f3rf3" %>%
  str_split(",") %>% # list containing one vector that contains splitted strings
  pluck(1) %>% #get the first list in outer list
  map_chr(str_trim) %>% # remove white spaces
  discard(~ .=="" | .=="NA") %>% # discard empty strings or "NA"
  keep(~ str_extract(., ".$") %in% 0:9) %>% # keep those ends with a digit
  discard(~ as.numeric(str_extract(.x, "[:digit:]+$")) < 5)
```

```
## [1] "absc61"     "sf22ve23"   "3rf3f344"   "rffr3$f446" "rf11"
```

```
# head_while and tail_while
sample(1:100) %>%
  head_while(~ . < 10)
```

```
## integer(0)
```

```
sample(1:100) %>%
  tail_while(~ . > 20)
```

```
## [1] 41 72 66 45 64 39 54
```

```
# run a function for a number of times
(m <- rerun(10, rnorm(10)) %>% # provides a list of 10 vectors each containing 10 iid
  reduce(cbind))
```

```
##                out         elt          elt         elt         elt         elt
```

```
##  [1,] -0.003693228 -0.53162947 -0.073265457 -2.06332037  0.06826259  1.03806324
##  [2,] -1.986603548  0.84963590  0.414526830 -0.05544542 -0.86876078  0.01307581
##  [3,]  0.696602032 -0.64030027 -1.796138323 -0.33507312 -1.24205509 -0.71267856
##  [4,]  0.594876404 -0.61543756  1.275858375 -1.67860144  1.07821182 -0.61331443
##  [5,]  0.400753246 -0.19539386 -0.326450814 -1.62401544 -0.55391058  0.76721377
##  [6,]  1.009947473 -0.06898662  0.103769141  0.87658993  1.00049719 -0.45891046
##  [7,] -0.077951068 -1.45528296  0.049289322 -0.31537083  0.60229613 -1.34330377
##  [8,]  1.581991306 -1.01644081  0.005218434  0.49351235  1.60952678  0.09937048
##  [9,] -1.036262955  0.17054258 -1.389764318 -0.19292095 -1.11671678 -1.05094434
## [10,] -0.019250413 -0.69521498  0.893593150 -1.57366617 -0.41254626 -0.81741506
##              elt         elt         elt         elt
##  [1,] -1.6332814 -2.33715213 -0.56569220  0.55177412
##  [2,]  0.6962374 -0.97263092  0.77852326 -0.32916842
##  [3,]  0.9164208 -0.26722464  0.47778509 -0.45675441
##  [4,]  0.8307660  1.65647952 -1.21299270  0.02959596
##  [5,] -0.6715974 -1.85845548  0.03036049  0.38028667
##  [6,] -0.8877525 -0.09709424  0.16933526  0.15103086
##  [7,]  0.1510978  0.63994928  2.64663106  0.79179134
##  [8,]  1.5778348 -0.14545980 -0.01550746 -0.05518850
##  [9,]  0.0140790 -0.23360764  2.82772359  1.79972244
## [10,]  0.3696307 -1.80915062 -0.15071502  1.51866287
```

```
colnames(m) <- 1:10 %>% map_chr (~ str_c("col", .))
as_tibble(m)
```

```
## # A tibble: 10 x 10
##        col1    col2     col3    col4    col5    col6    col7    col8    col9
##       <dbl>   <dbl>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1 -0.00369 -0.532  -0.0733  -2.06    0.0683  1.04   -1.63   -2.34   -0.566
##  2 -1.99     0.850   0.415   -0.0554 -0.869   0.0131  0.696  -0.973   0.779
##  3  0.697   -0.640  -1.80    -0.335  -1.24   -0.713   0.916  -0.267   0.478
##  4  0.595   -0.615   1.28    -1.68    1.08   -0.613   0.831   1.66   -1.21
##  5  0.401   -0.195  -0.326   -1.62   -0.554   0.767  -0.672  -1.86    0.0304
##  6  1.01    -0.0690  0.104    0.877   1.00   -0.459  -0.888  -0.0971  0.169
##  7 -0.0780  -1.46    0.0493  -0.315   0.602  -1.34    0.151   0.640   2.65
##  8  1.58    -1.02    0.00522  0.494   1.61    0.0994  1.58   -0.145  -0.0155
##  9 -1.04     0.171  -1.39    -0.193  -1.12   -1.05    0.0141 -0.234   2.83
## 10 -0.0193  -0.695   0.894   -1.57   -0.413  -0.817   0.370  -1.81   -0.151
## # ... with 1 more variable: col10 <dbl>
```

```
# imap is indexed map
rerun(.n = 10, rnorm(10)) %>%
  imap_dfr(~ tibble(run = .y,
                    mean = mean(.x),
                    sd = sd(.x),
                    median = median(.x)))
```

```
## # A tibble: 10 x 4
##      run    mean    sd  median
##    <int>   <dbl> <dbl>   <dbl>
##  1     1  0.220  1.04   0.231
##  2     2 -0.308  0.860 -0.265
##  3     3 -0.0771 0.517  0.0500
##  4     4 -0.158  0.602 -0.0669
##  5     5 -0.0602 0.899 -0.0134
##  6     6  0.268  1.02   0.138
```

```
##  7     7 -0.136  1.24  -0.0488
##  8     8  0.409  0.881  0.290
##  9     9  0.559  1.02   0.894
## 10    10  0.0246 1.13   0.403
```

```
suppressWarnings(suppressMessages(library(tidyverse)))
library(tidyverse)
# reduce /  reduce_right is R's foldleft / foldRight

# -----------  foldLeft(List[Int]())((acc, x) => acc :+ (x + 1)) -----------
# Note #1: The First argument of the function is always zero of monoid.
# you can set zero of monoid using ".init" argument of reduce.
# If .init is not provided then first element of the list is used as zero.
# Note '.init = NULL' forces reduce to use NULL (i.e. an empty list) as
# zero of the monoid and pass it as the first argument to the provided function
# rather than using .x[[1]].
# reduce passes elements of the collection as function's second argument.

list(1,2,3,4,5) %>%
  reduce(~append (.x, .y + 1), .init = NULL)
```

```
## [1] 2 3 4 5 6
```

```
list(1,2,3,4,5) %>%
  reduce_right (~append (.x, .y + 1), .init = NULL)
```

```
## Warning: `reduce_right()` is soft-deprecated as of purrr 0.3.0.
## Please use the new `.dir` argument of `reduce()` instead.
##
##   # Before:
##   reduce_right(1:3, f)
##
##   # After:
##   reduce(1:3, f, .dir = "backward")  # New algorithm
##   reduce(rev(1:3), f)                # Same algorithm as reduce_right()
##
## This warning is displayed once per session.
```

```
## [1] 6 5 4 3 2
```

```
# reduce to find intersection between multiple lists



# use reduce to append multiple data frames
df1 <- data.frame(Customer_ID = c(1, 2, 3), Name = c("John", "Sue", "Ann"))
df2 <- data.frame(Customer_ID = c(4, 5, 6), Name = c("Joe", "Suzy", "Annable"))

list(df1, df2) %>%
  reduce(rbind)
```

```
##   Customer_ID    Name
## 1           1    John
## 2           2     Sue
## 3           3     Ann
## 4           4     Joe
## 5           5    Suzy
```

```
## 6           6 Annable
```

```
# accumulate does exactly the same thing as reduce but it also returns
# accumulated value on each loop, provided .init is accessible by index
list(df1, df2) %>%
  accumulate(rbind)
```

```
## [[1]]
##   Customer_ID Name
## 1           1 John
## 2           2  Sue
## 3           3  Ann
##
## [[2]]
##   Customer_ID    Name
## 1           1    John
## 2           2     Sue
## 3           3     Ann
## 4           4     Joe
## 5           5    Suzy
## 6           6 Annable
```

```
# ----------- merge (join, left join, right join and ntural join) ----------
# First remember what "all" option of "merge" does:
# all = FALSE (default value) inner join where columns with the same name of
#                             associated tables will appear once only.
# all.x = TRUE gives a left (outer) join,
# all.y = TRUE a right (outer) join,
# (all = TRUE) a (full) outer join.
# DBMSes do not match NULL records, equivalent to incomparables = NA in R.

#  More data for merge
df1 <- data.frame(Customer_ID = c(1, 2, 3), Name = c("John", "Sue", "Ann"))
df2 <- data.frame(Customer_ID = c(1, 3), Year_First_Order = c(2011, 2017))
df3 <- data.frame(Customer_ID = c(1, 2, 3),
                  Date_Last_Order = c("2017-03-03", "2014-03-01", "2017-05-30"),
                  No_Items_Last_Order = c(3, 1, 1),
                  Total_Amount_Last_Order = c(49, 25,25))
df4 <- data.frame(Customer_ID = c(2, 3), Interested_In_Promo = c(TRUE, FALSE))

merge(df1 , df2, by="Customer_ID")
```

```
##   Customer_ID Name Year_First_Order
## 1           1 John             2011
## 2           3  Ann             2017
```

```
merge(df1 , df2, by="Customer_ID", all.x = T)
```

```
##   Customer_ID Name Year_First_Order
## 1           1 John             2011
## 2           2  Sue               NA
## 3           3  Ann             2017
```

```
# now we use reduce to join the dataframes
list(df1,df2,df3,df4) %>%
  reduce(~ merge(.x, .y, all.x = T))
```

```
##   Customer_ID Name Year_First_Order Date_Last_Order No_Items_Last_Order
## 1           1 John             2011      2017-03-03                   3
## 2           2  Sue               NA      2014-03-01                   1
## 3           3  Ann             2017      2017-05-30                   1
##   Total_Amount_Last_Order Interested_In_Promo
## 1                      49                  NA
## 2                      25                TRUE
## 3                      25               FALSE
```

```
# another example of reduce : Sums of matrix powers
# we need package expm to do matrix power
library(expm)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
##
## Attaching package: 'expm'
```

```
## The following object is masked from 'package:Matrix':
##
##     expm
```

```
(m <- rbind(c(0.9, 0.1), c(1, 0)))
```

```
##      [,1] [,2]
## [1,]  0.9  0.1
## [2,]  1.0  0.0
```

```
1:20 %>%
  map(~ (m %^% .)) %>%
  reduce(~(.x+.y), .init = rbind(c(0,0),c(0.0)))
```

```
##          [,1]     [,2]
## [1,] 18.17355 1.826446
## [2,] 18.26446 1.735537
```

```
suppressWarnings(suppressMessages(library(tidyverse)))
library(tidyverse)
## A general-purpose adder:
add <- function(x) Reduce("+", x)
add(list(1, 2, 3))
```

```
## [1] 6
```

```
## Like sum(), but can also used for adding matrices etc., as it will
## use the appropriate '+' method in each reduction step.
## More generally, many generics meant to work on arbitrarily many
## arguments can be defined via reduction:
FOO <- function(...) Reduce(FOO2, list(...))
FOO2 <- function(x, y) UseMethod("FOO2")
## FOO() methods can then be provided via FOO2() methods.
```

```
## A general-purpose cumulative adder:
cadd <- function(x) Reduce("+", x, accumulate = TRUE)
cadd(seq_len(7))
```

```
## [1]  1  3  6 10 15 21 28
```

```
## A simple function to compute continued fractions:
cfrac <- function(x) Reduce(function(u, v) u + 1 / v, x, right = TRUE)
## Continued fraction approximation for pi:
cfrac(c(3, 7, 15, 1, 292))
```

```
## [1] 3.141593
```

```
## Continued fraction approximation for Euler's number (e):
cfrac(c(2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8))
```

```
## [1] 2.718282
```

```
## Iterative function application:
Funcall <- function(f, ...) f(...)
## Compute log(exp(acos(cos(0))))
Reduce(Funcall, list(log, exp, acos, cos), 0, right = TRUE)
```

```
## [1] 0
```

```
## n-fold iterate of a function, functional style:
Iterate <- function(f, n = 1)
    function(x) Reduce(Funcall, rep.int(list(f), n), x, right = TRUE)
## Continued fraction approximation to the golden ratio:
Iterate(function(x) 1 + 1 / x, 30)(1)
```

```
## [1] 1.618034
```

```
## which is the same as
cfrac(rep.int(1, 31))
```

```
## [1] 1.618034
```

```
## Computing square root approximations for x as fixed points of the
## function t |-> (t + x / t) / 2, as a function of the initial value:
asqrt <- function(x, n) Iterate(function(t) (t + x / t) / 2, n)
asqrt(2, 30)(10) # Starting from a positive value => +sqrt(2)
```

```
## [1] 1.414214
```

```
asqrt(2, 30)(-1) # Starting from a negative value => -sqrt(2)
```

```
## [1] -1.414214
```

```
## A list of all functions in the base environment:
funs <- Filter(is.function, sapply(ls(baseenv()), get, baseenv()))
## Functions in base with more than 10 arguments:
names(Filter(function(f) length(formals(f)) > 10, funs))
```

```
## [1] "format.default"   "formatC"          "library"          "merge.data.frame"
## [5] "prettyNum"        "scan"             "source"           "system2"
```

```
## Number of functions in base with a '...' argument:
length(Filter(function(f)
              any(names(formals(f)) %in% "..."),
              funs))
```

```
## [1] 414
```

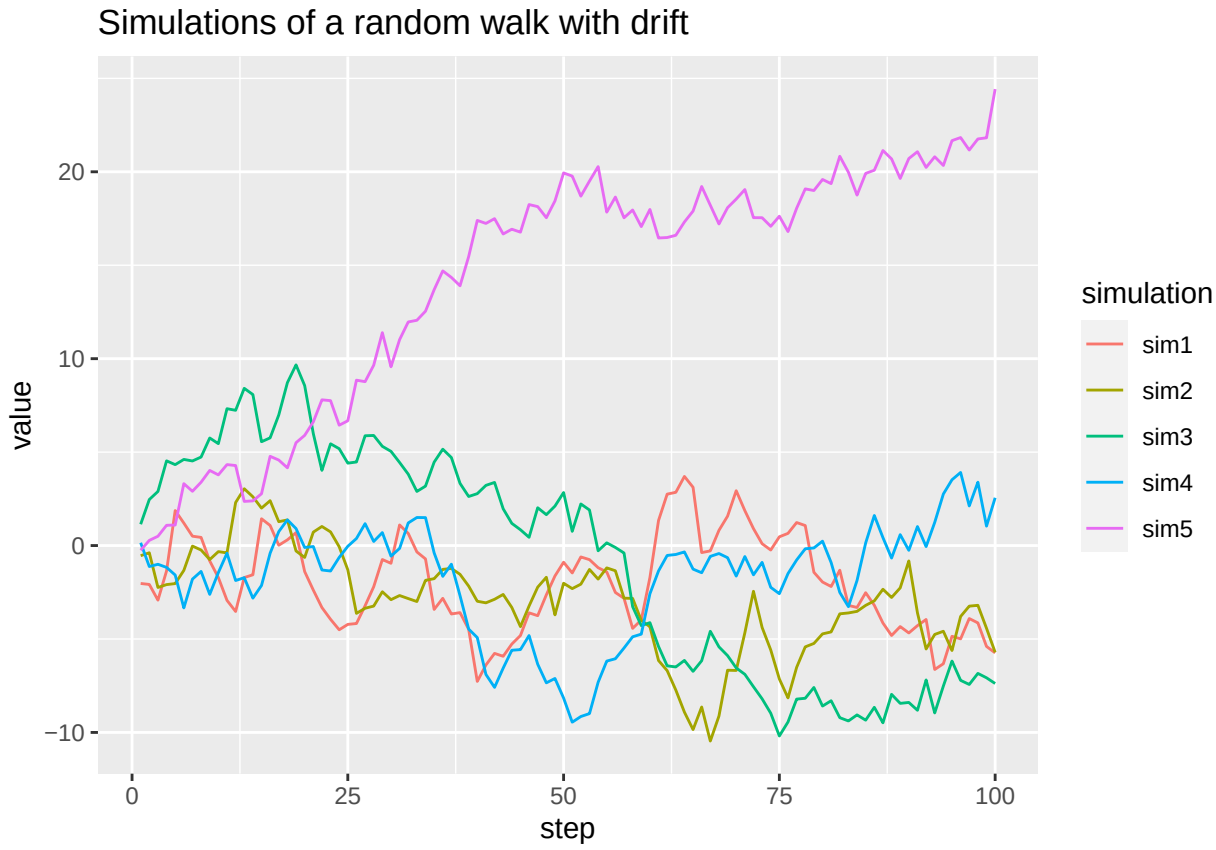
```
## Find all objects in the base environment which are *not* functions:
Filter(Negate(is.function),  sapply(ls(baseenv()), get, baseenv()))
```

```
## $F
## [1] FALSE
##
## $letters
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
##
## $LETTERS
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
##
## $month.abb
##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
##
## $month.name
##  [1] "January"   "February"  "March"     "April"     "May"       "June"
##  [7] "July"      "August"    "September" "October"   "November"  "December"
##
## $pi
## [1] 3.141593
##
## $R.version
##                _
## platform       x86_64-apple-darwin15.6.0
## arch           x86_64
## os             darwin15.6.0
## system         x86_64, darwin15.6.0
## status
## major          3
## minor          6.2
## year           2019
## month          12
## day            12
## svn rev        77560
## language       R
## version.string R version 3.6.2 (2019-12-12)
## nickname       Dark and Stormy Night
##
## $R.version.string
## [1] "R version 3.6.2 (2019-12-12)"
##
## $T
## [1] TRUE
##
## $version
##                _
## platform       x86_64-apple-darwin15.6.0
## arch           x86_64
## os             darwin15.6.0
## system         x86_64, darwin15.6.0
```

```
## status
## major          3
## minor          6.2
## year           2019
## month          12
## day            12
## svn rev        77560
## language       R
## version.string R version 3.6.2 (2019-12-12)
## nickname       Dark and Stormy Night
```

```
suppressWarnings(suppressMessages(library(tidyverse)))
library(tidyverse)

rerun(5, rnorm(100)) %>%
  set_names(paste0("sim", 1:5)) %>%
  map(~ accumulate(., ~ .05 + .x + .y)) %>%
  map_dfr(~ tibble(value = .x, step = 1:100), .id = "simulation") %>%
  ggplot(aes(x = step, y = value)) +
    geom_line(aes(color = simulation)) +
    ggtitle("Simulations of a random walk with drift")
```




```
# We use rep.int as rep is primitive
rep.int(12,5)
```

```
## [1] 12 12 12 12 12
```

```
vrep <- Vectorize(rep.int)
vrep(1:4, 4:1)
```

```
## [[1]]
## [1] 1 1 1 1
##
## [[2]]
## [1] 2 2 2
##
## [[3]]
## [1] 3 3
##
## [[4]]
## [1] 4
```

```
vrep(times = 1:4, x = 4:1)
```

```
## [[1]]
## [1] 4
##
## [[2]]
## [1] 3 3
##
## [[3]]
## [1] 2 2 2
##
## [[4]]
## [1] 1 1 1 1
```

```
vrep <- Vectorize(rep.int, "times")
vrep(times = 1:4, x = 42)
```

```
## [[1]]
## [1] 42
##
## [[2]]
## [1] 42 42
##
## [[3]]
## [1] 42 42 42
##
## [[4]]
## [1] 42 42 42 42
```

```
f <- function(x = 1:3, y) c(x, y)
vf <- Vectorize(f, SIMPLIFY = FALSE)
f(1:3, 1:3)
```

```
## [1] 1 2 3 1 2 3
```

```
vf(1:3, 1:3)
```

```
## [[1]]
## [1] 1 1
##
## [[2]]
## [1] 2 2
##
```

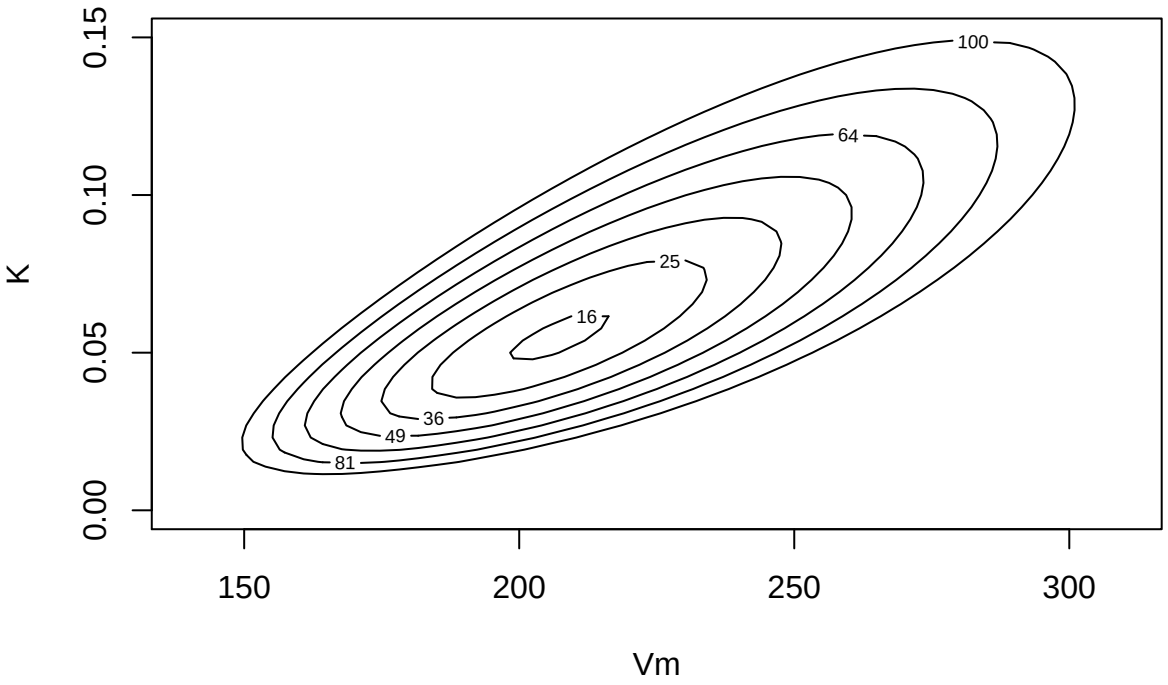
```
## [[3]]
## [1] 3 3
```

```
vf(y = 1:3) # Only vectorizes y, not x
```

```
## [[1]]
## [1] 1 2 3 1
##
## [[2]]
## [1] 1 2 3 2
##
## [[3]]
## [1] 1 2 3 3
```

```
# Nonlinear regression contour plot, based on nls() example
require(graphics)
SS <- function(Vm, K, resp, conc) {
    pred <- (Vm * conc)/(K + conc)
    sum((resp - pred)^2 / pred)
}
vSS <- Vectorize(SS, c("Vm", "K"))
Treated <- subset(Puromycin, state == "treated")

Vm <- seq(140, 310, length.out = 50)
K <- seq(0, 0.15, length.out = 40)
SSvals <- outer(Vm, K, vSS, Treated$rate, Treated$conc)
contour(Vm, K, SSvals, levels = (1:10)^2, xlab = "Vm", ylab = "K")
```




```
# combn() has an argument named FUN
combnV <- Vectorize(function(x, m, FUNV = NULL) combn(x, m, FUN = FUNV),
                    vectorize.args = c("x", "m"))
combnV(4, 1:4)
```

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

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
##
## [[2]]
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    1    1    2    2    3
## [2,]    2    3    4    3    4    4
##
## [[3]]
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    1    2
## [2,]    2    2    3    3
## [3,]    3    4    4    4
##
## [[4]]
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
## [4,]    4
```

```
combnV(4, 1:4, sum)
```

```
## [[1]]
## [1] 1 2 3 4
##
## [[2]]
## [1] 3 4 5 5 6 7
##
## [[3]]
## [1] 6 7 8 9
##
## [[4]]
## [1] 10
```

```
suppressWarnings(suppressMessages(library(tidyverse)))
library(tidyverse)

wage.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Wage.csv",
                   header=T, stringsAsFactors = F, na.strings = "?")
wage.df.original = tibble(wage.df)
(wage.df = tibble(wage.df))
```

```
## # A tibble: 3,000 x 12
##     year   age sex   maritl race  education region jobclass health health_ins
##    <int> <int> <chr> <chr>  <chr> <chr>     <chr>  <chr>    <chr>  <chr>
##  1  2006    18 1. M~ 1. Ne~ 1. W~ 1. < HS ~ 2. Mi~ 1. Indu~ 1. <=~ 2. No
##  2  2004    24 1. M~ 1. Ne~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 2. No
##  3  2003    45 1. M~ 2. Ma~ 1. W~ 3. Some ~ 2. Mi~ 1. Indu~ 1. <=~ 1. Yes
##  4  2003    43 1. M~ 2. Ma~ 3. A~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  5  2005    50 1. M~ 4. Di~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  6  2008    54 1. M~ 2. Ma~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  7  2009    44 1. M~ 2. Ma~ 4. O~ 3. Some ~ 2. Mi~ 1. Indu~ 2. >=~ 1. Yes
##  8  2008    30 1. M~ 1. Ne~ 3. A~ 3. Some ~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  9  2006    41 1. M~ 1. Ne~ 2. B~ 3. Some ~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
```

```
## 10  2004    52 1. M~ 2. Ma~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## # ... with 2,990 more rows, and 2 more variables: logwage <dbl>, wage <dbl>
```

```
# remove empty characters and NA helper
remove.empty.characters <- function(df)
  df %>%
    select_all %>%
    filter_if(is.character, any_vars(!is.na(.) & trimws(.) != ""))

# Next remove leading and trailing spaces from all elements in character columns
trim.f <- function(col, na.rm = F) {
  isNA <- !reduce(col, ~ (is.na(.x) & is.na(.y)))
  if (na.rm && isNA)
    unlist(map(col, ~ (if (is.na(.x)) "" else .x) ),use.names = F)
  else trimws(col, which = c("both")) # leading and trailing spaces
}

trim.spaces <- function(df)
  (df %>%
    mutate_if(is.character, trim.f, na.rm = T))

# Finally convert character columns to factor

char.to.fctor <- function(df)
  df %>%
    mutate_if(is.character, ~ factor(.x, levels = (.x %>% table() %>% names())))


(wage.df <-
  wage.df %>%
  na.omit() %>%
  trim.spaces() %>%
  remove.empty.characters() %>%
  char.to.fctor())
```

```
## # A tibble: 3,000 x 12
##     year   age sex   maritl race  education region jobclass health health_ins
##    <int> <int> <fct> <fct>  <fct> <fct>     <fct>  <fct>    <fct>  <fct>
##  1  2006    18 1. M~ 1. Ne~ 1. W~ 1. < HS ~ 2. Mi~ 1. Indu~ 1. <=~ 2. No
##  2  2004    24 1. M~ 1. Ne~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 2. No
##  3  2003    45 1. M~ 2. Ma~ 1. W~ 3. Some ~ 2. Mi~ 1. Indu~ 1. <=~ 1. Yes
##  4  2003    43 1. M~ 2. Ma~ 3. A~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  5  2005    50 1. M~ 4. Di~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  6  2008    54 1. M~ 2. Ma~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  7  2009    44 1. M~ 2. Ma~ 4. O~ 3. Some ~ 2. Mi~ 1. Indu~ 2. >=~ 1. Yes
##  8  2008    30 1. M~ 1. Ne~ 3. A~ 3. Some ~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  9  2006    41 1. M~ 1. Ne~ 2. B~ 3. Some ~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## 10  2004    52 1. M~ 2. Ma~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## # ... with 2,990 more rows, and 2 more variables: logwage <dbl>, wage <dbl>
```

- References :
  - Advanced R second edition by Hadley Wickham
  - http://www.rebeccabarter.com/blog/2019-08-19_purrr/
  - https://suzanbaert.netlify.app/2018/01/dplyr-tutorial-1/
  - https://suzanbaert.netlify.app/2018/01/dplyr-tutorial-2/

– https://suzanbaert.netlify.app/2018/01/dplyr-tutorial-3/
– https://suzanbaert.netlify.app/2018/01/dplyr-tutorial-4/