More Tips and Tricks

Abhinav Anand, IIMB

2019/06/18

Scripts in R

Scripts are files (ending in `.R`) that contain sequences of instructions that we can command a machine to follow. Consier a typical empirical project: we need to read data stored in some type of file; then clean and tidy it, then process it and use descriptive statistics and plots to delineate its features etc. All this can be achieved by storing a sequence of instructions in a script file. Data can be read using the the `read_csv()` function, processing can be done using the package `dplyr` etc.

Linear regression in R

Generally a linear model takes the following form:

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_m x_m + u$$

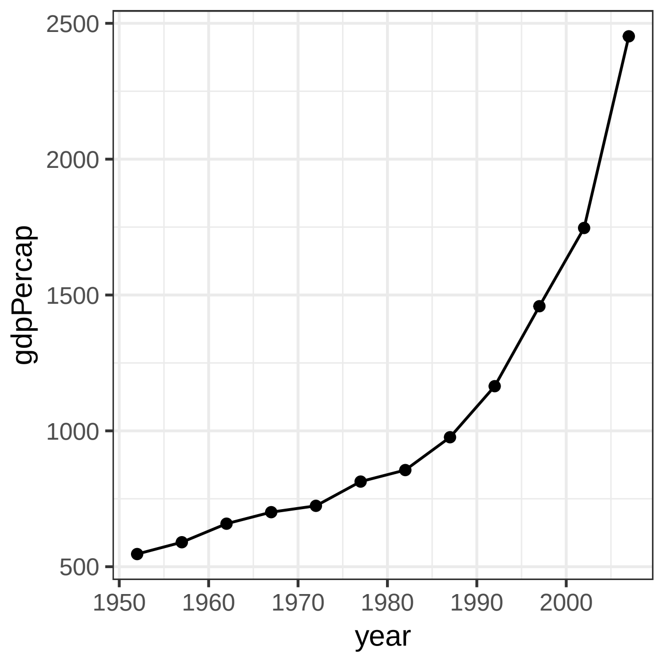
where $u_{n\times 1}$ is the error term. This setup corresponds to an overdetermined linear system of equations leading to a least squares solution:

$$\hat{\beta} = (X^\top X)^{-1} X^\top y$$

where the explanatory matrix $X_{m\times n}$ contains independent variables $x_1, \ldots, x_m$ as column vectors of size $n \times 1$.

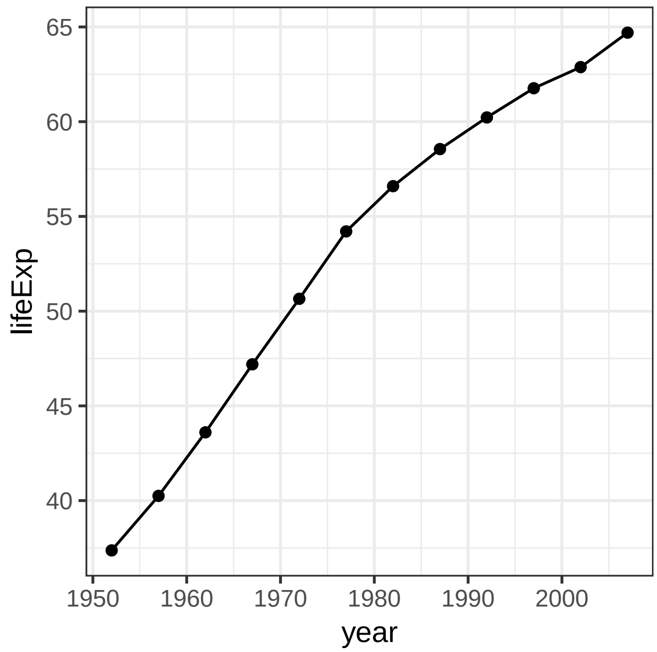
One of the strengths of R is the flexibility and support it offers for linear regression modeling. In order to illustrate it more fully, let us consider data for India in the `gapminder` dataset.

```
data_Ind <- gapminder::gapminder %>%
  dplyr::filter(country == "India")

ggplot(data_Ind, aes(year, gdpPercap)) +
  geom_point() +
  geom_line()
```



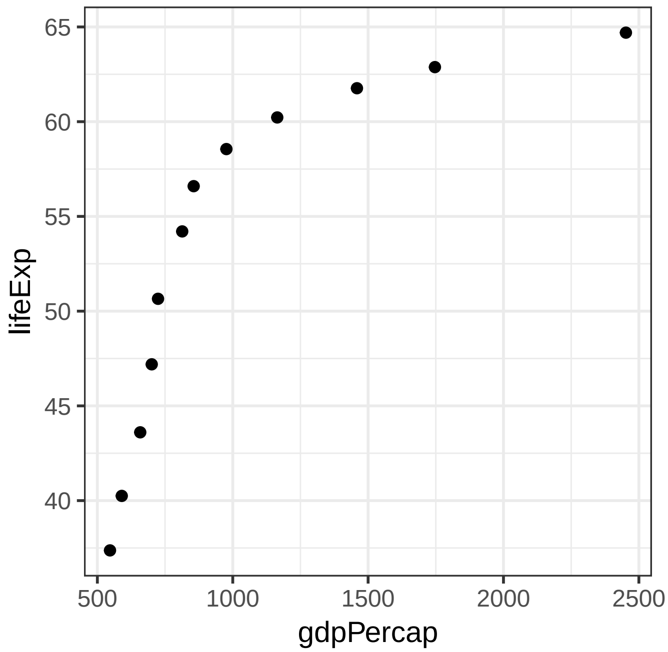
We see that there has been a large increase in GDP per capita in India. A similar trend is observed for life expectancy:

```
ggplot(data_Ind, aes(year, lifeExp)) +
  geom_point() +
  geom_line()
```



What about the relationship between the two? For example, (all else equal) does GDP per capita of India explain the life expectancy trends observed?

```
ggplot(data_Ind, aes(gdpPercap, lifeExp)) +
  geom_point()
```



This suggests that the two variables share a positive relation. We can try to check this by means of a linear regression in the following way:

$$\text{life exp} = \beta_0 + \beta_1(\text{gdp percap}) + u$$

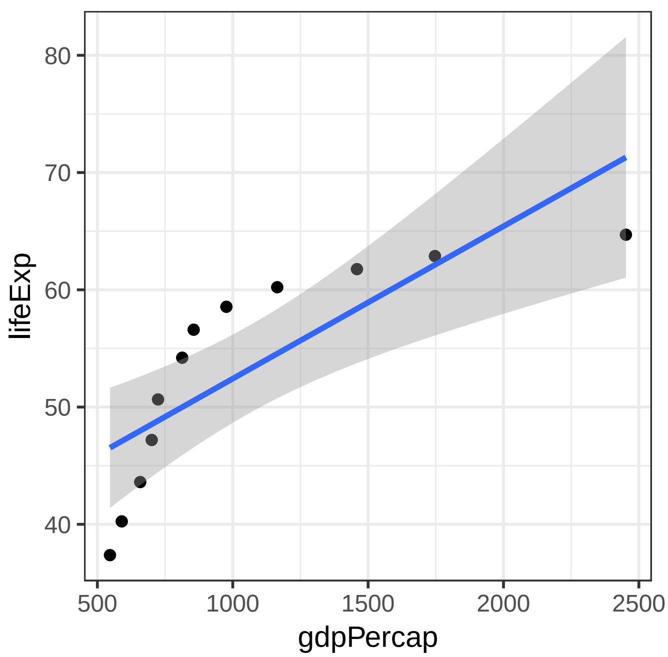
In order to implement this step in R is via the following:

```
lm_formula <- lifeExp ~ gdpPercap

lm_life_gdppc <- lm(data = data_Ind, formula = lm_formula)

summary(lm_life_gdppc)
```

```
##
## Call:
## lm(formula = lm_formula, data = data_Ind)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -9.155 -4.931   1.284   4.576   6.437
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 39.423336   3.659432  10.773    8e-07 ***
## gdpPercap    0.012998   0.003075   4.227  0.00175 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.816 on 10 degrees of freedom
## Multiple R-squared:  0.6411, Adjusted R-squared:  0.6052
## F-statistic: 17.86 on 1 and 10 DF,  p-value: 0.001753
```

```r
ggplot(data_Ind, aes(gdpPercap, lifeExp)) +
  geom_point() +
  geom_smooth(method = "lm")
```



What is this object `lm_life_gdppc`? What is its structure? We can quickly check by accessing its contents:

```
names(lm_life_gdppc)
```

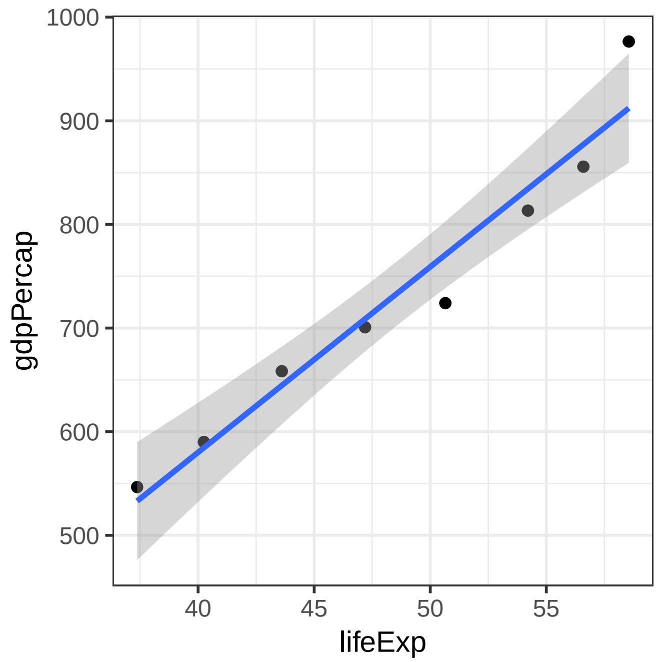
```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

What about some subset of data, say the period before 1990?

```
lm(filter(data_Ind, year <= 1990), formula = lm_formula) %>%
  summary()
```

```
##
## Call:
## lm(formula = lm_formula, data = filter(data_Ind, year <= 1990))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8626 -1.0747 -0.1943  1.4541  2.5804
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  9.80149    3.83775   2.554   0.0433 *
## gdpPercap    0.05286    0.00515  10.264 4.99e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.945 on 6 degrees of freedom
## Multiple R-squared:  0.9461, Adjusted R-squared:  0.9371
## F-statistic: 105.3 on 1 and 6 DF,  p-value: 4.992e-05
```

```r
ggplot(filter(data_Ind, year <= 1990), aes(lifeExp, gdpPercap)) +
  geom_point() +
  geom_smooth(method = "lm")
```



Nonlinear relationships

The plot between the dependent and independent variable suggest a nonlinear relationship. Can we test this simply? Let's consider the following modification:

$$\text{life exp} = \beta_0 + \beta_1(\text{gdp percap})^2 + u$$

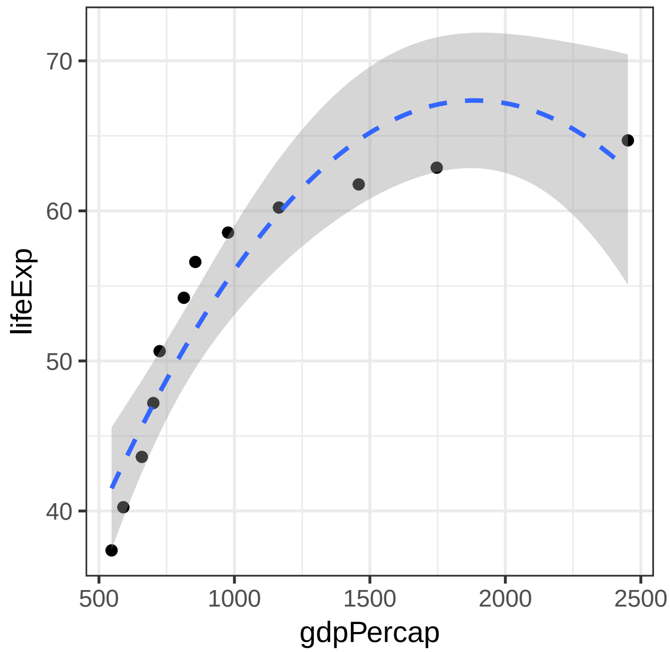
In general, R can accommodate independent variables involving mathematical operators in a regression equation with the function `I()`.

```r
lm_formula_quad <- lifeExp ~ I(gdpPercap)^2

lm_life_gdppc_quad <- lm(data = data_Ind, formula = lm_formula_quad)
```

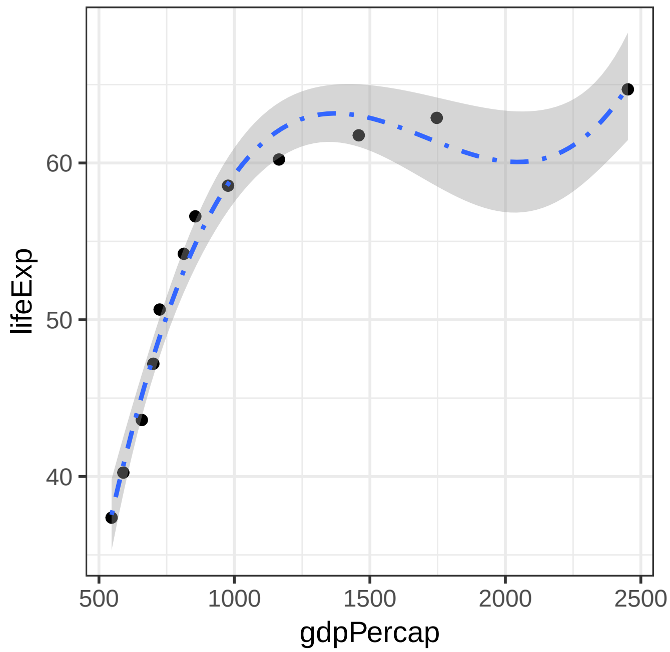
```r
summary(lm_life_gdppc_quad)
```

```
##
## Call:
## lm(formula = lm_formula_quad, data = data_Ind)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -9.155 -4.931  1.284  4.576  6.437
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  39.423336   3.659432  10.773    8e-07 ***
## I(gdpPercap)  0.012998   0.003075   4.227  0.00175 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.816 on 10 degrees of freedom
## Multiple R-squared:  0.6411, Adjusted R-squared:  0.6052
## F-statistic: 17.86 on 1 and 10 DF,  p-value: 0.001753
```

```r
ggplot(data_Ind, aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
  stat_smooth(method = "lm",
              formula = y ~ poly(x, 2), #polynomial order 2
              size = 0.8,
              linetype = "dashed"
              )
```




```
# What about higher order polynomials?
ggplot(data_Ind, aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
  stat_smooth(method = "lm",
              formula = y ~ poly(x, 3), #polynomial order 3
              size = 0.8,
              linetype = "dotdash"
              )
```



Are visually better fits also evidence of better underlying models? This is a hard question to answer in general—all else equal we prefer models that are parsimonious (have fewer explanatory variables).

Functional Programming in R

Another very powerful feature of R is its support for functional programming, which in general, involves applying functions to arrays, dataframes, lists etc.

For example, how should one compute the mean across rows of a matrix?

```
df <- data.frame(C_1 = rnorm(10, 0, 1),
                 C_2 = rnorm(10, 1, 2),
                 C_3 = rnorm(10, 2, 3)
                 )


head(df)
```

```
##          C_1        C_2        C_3
## 1  0.3644709  1.3382598 -1.2066315
## 2  0.8944397  1.6764120  3.2312982
## 3  1.9975643 -0.7821961 -1.7439697
## 4  1.3048047 -0.9154783  2.3508948
## 5 -1.0823374 -1.6868917 -0.8635651
## 6  1.3252254 -1.0091568  3.7084433
```

```r
# One way to solve the problem
rmean_1 <- rowMeans(df)
print(rmean_1)
```

```
##  [1]  0.16536640  1.93404996 -0.17620049  0.91340708 -1.21093139
##  [6]  1.34150398  1.40577541  1.88427922  0.22121314  0.06221691
```

```r
# Another more 'functional' way
func_mean <- function(vec)
{
  return(mean(vec, na.rm = T))
}

# Apply function on rows
rmean_2 <- apply(df, 1, func_mean)
print(rmean_1)
```

```
##  [1]  0.16536640  1.93404996 -0.17620049  0.91340708 -1.21093139
##  [6]  1.34150398  1.40577541  1.88427922  0.22121314  0.06221691
```

```r
# Apply function on columns
rmean_3 <- apply(df, 2, func_mean)
print(rmean_3)
```

```
##         C_1         C_2         C_3
##  0.54442698 -0.04173822  1.45951531
```

Note how to use the `apply()` function. We `apply()` the function over rows or columns or other dimensions. In general that's the philosophy of the `apply()` family of functions, which includes functions `lapply()` (list-apply) and `sapply()` (simplify-apply) etc. The function `lapply` returns a list and `sapply` a vector (if possible). In both cases the first argument is a list (or dataframe) and the second argument is the name of a function.

What is a list? It's essentially a more general version of a dataframe and can contain not only dissimilar data types but also, say dataframes within them.

```
temp_list <- list(a = runif(10),
                  b = "Happy birthday",
                  c = data.frame(x = rnorm(10, 0, 1)),
                  d = sample(letters, 7, replace = TRUE)
                  )
str(temp_list) #structure of the list
```

```
## List of 4
##  $ a: num [1:10] 0.231 0.126 0.15 0.774 0.69 ...
##  $ b: chr "Happy birthday"
##  $ c:'data.frame':   10 obs. of  1 variable:
##   ..$ x: num [1:10] 0.116 -1.248 -1.682 -0.577 1.055 ...
##  $ d: chr [1:7] "u" "l" "c" "s" ...
```

```
# lapply() is used to apply the same function to each
# "cell" of the list
lapply(temp_list, is.numeric) #check if each cell is numeric
```

```
## $a
## [1] TRUE
##
## $b
## [1] FALSE
```

```
##
## $c
## [1] FALSE
##
## $d
## [1] FALSE
```

```
# contrast with sapply()
sapply(temp_list, is.numeric)
```

```
##     a     b     c     d
##  TRUE FALSE FALSE FALSE
```

The map() family from purrr

The `map` function does the exact same operation as `apply` but is consistent with the output format type. For example `map()` returns a list, `map_dbl()` returns a double type vector, `map_int()` returns an integer type vector etc. As with `read_csv`, the `map` family improves upon the base R code by being faster and more consistent.

```
map(df, mean)
```

```
## $C_1
## [1] 0.544427
##
## $C_2
## [1] -0.04173822
##
## $C_3
## [1] 1.459515
```

```r
map_dbl(df, median)
```

```
##        C_1        C_2        C_3
##  0.6673511 -0.4732725  1.2362129
```

```r
# Also compare this
z <- list(x = 1:3, y = 4:5)
map_int(z, length) #names are preserved
```

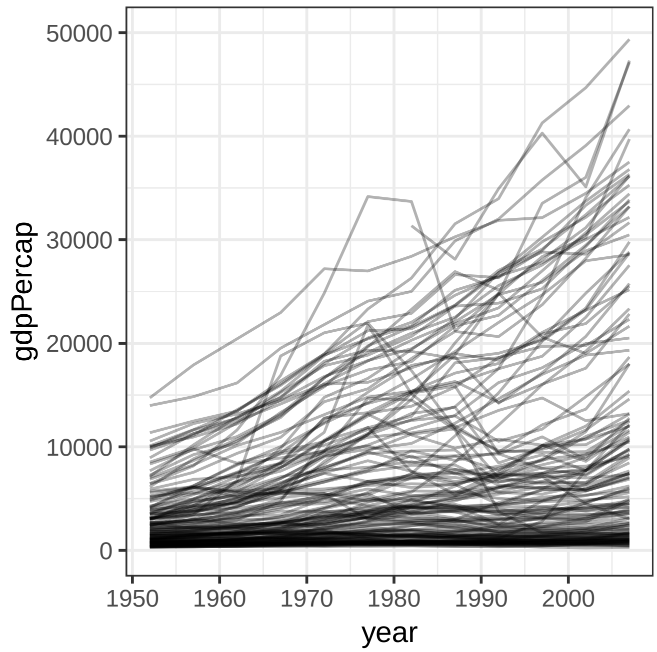
```
## x y
## 3 2
```

Advanced tricks: Nesting and multiple models

We return to the `gapminder` dataset to look at GDP per capita around the world.

```r
gapminder %>%
  ggplot(., aes(year, gdpPercap, group = country)) +
  geom_line(alpha = 0.3) +
  ylim(0, 50000)
```

```
## Warning: Removed 6 rows containing missing values (geom_path).
```



Overall, it looks like GDP per capita has been steadily improving though some countries see some declines.

Nested data frame

This is best understood by means of examples.

```
head(gapminder)
```

```
## # A tibble: 6 x 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

```r
# What nesting does
nest_country <- gapminder %>%
  dplyr::group_by(country, continent) %>%
  tidyr::nest()

nest_country
```

```
## # A tibble: 142 x 3
##    country     continent data
##    <fct>       <fct>     <list>
##  1 Afghanistan Asia      <tibble [12 x 4]>
##  2 Albania     Europe    <tibble [12 x 4]>
##  3 Algeria     Africa    <tibble [12 x 4]>
##  4 Angola      Africa    <tibble [12 x 4]>
##  5 Argentina   Americas  <tibble [12 x 4]>
##  6 Australia   Oceania   <tibble [12 x 4]>
##  7 Austria     Europe    <tibble [12 x 4]>
##  8 Bahrain     Asia      <tibble [12 x 4]>
##  9 Bangladesh  Asia      <tibble [12 x 4]>
## 10 Belgium     Europe    <tibble [12 x 4]>
## # ... with 132 more rows
```

```r
# Accessing country-level data
nest_country$data[[1]] #Afghanistan
```

```
## # A tibble: 12 x 4
##     year lifeExp      pop gdpPercap
##    <int>   <dbl>    <int>     <dbl>
##  1  1952    28.8  8425333      779.
##  2  1957    30.3  9240934      821.
##  3  1962    32.0 10267083      853.
##  4  1967    34.0 11537966      836.
```

```
##  5  1972    36.1 13079460      740.
##  6  1977    38.4 14880372      786.
##  7  1982    39.9 12881816      978.
##  8  1987    40.8 13867957      852.
##  9  1992    41.7 16317921      649.
## 10  1997    41.8 22227415      635.
## 11  2002    42.1 25268405      727.
## 12  2007    43.8 31889923      975.
```

Now we apply previously discussed ideas from functional programming to such nested dataframes. We want to check if the trend has been rising for each country. This can be achieved by applying a trend-computing function to each dataframe in the nested object.

```r
# Trend computing function for GDP per capita
lm_gdppc_year <- function(data_frame)
{
  temp <- lm(data = data_frame,
             formula = gdpPercap ~ year) #trend
  return((temp))
}

# Update the nested dataframe by incorporating trends
nest_country <- nest_country %>%
  dplyr::mutate(trend = map(data, lm_gdppc_year))
# Equivalently: map(nest_country$data, lm_gdppc_year)

nest_country$trend[[1]] #Afghanistan
```

```
##
## Call:
## lm(formula = gdpPercap ~ year, data = data_frame)
```

```
##
## Coefficients:
## (Intercept)         year
##   1674.8134      -0.4406
```

```r
# We can also do the standard procedures
nest_country %>%
  dplyr::filter(continent == 'Asia')
```

```
## # A tibble: 33 x 4
##    country          continent data              trend
##    <fct>            <fct>     <list>            <list>
##  1 Afghanistan      Asia      <tibble [12 x 4]> <lm>
##  2 Bahrain          Asia      <tibble [12 x 4]> <lm>
##  3 Bangladesh       Asia      <tibble [12 x 4]> <lm>
##  4 Cambodia         Asia      <tibble [12 x 4]> <lm>
##  5 China            Asia      <tibble [12 x 4]> <lm>
##  6 Hong Kong, China Asia      <tibble [12 x 4]> <lm>
##  7 India            Asia      <tibble [12 x 4]> <lm>
##  8 Indonesia        Asia      <tibble [12 x 4]> <lm>
##  9 Iran             Asia      <tibble [12 x 4]> <lm>
## 10 Iraq             Asia      <tibble [12 x 4]> <lm>
## # ... with 23 more rows
```

Unnesting

The `unnest()` function undoes the nesting:

```r
tidyr::unnest(nest_country, data)
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
```

```
##    1 Afghanistan Asia       1952     28.8  8425333      779.
##    2 Afghanistan Asia       1957     30.3  9240934      821.
##    3 Afghanistan Asia       1962     32.0 10267083      853.
##    4 Afghanistan Asia       1967     34.0 11537966      836.
##    5 Afghanistan Asia       1972     36.1 13079460      740.
##    6 Afghanistan Asia       1977     38.4 14880372      786.
##    7 Afghanistan Asia       1982     39.9 12881816      978.
##    8 Afghanistan Asia       1987     40.8 13867957      852.
##    9 Afghanistan Asia       1992     41.7 16317921      649.
##   10 Afghanistan Asia       1997     41.8 22227415      635.
## # ... with 1,694 more rows
```

Evaluating model fits

The `glance()` function from the package `broom` (included in `tidyverse`) is useful

```r
nest_country %>%
  dplyr::mutate(model_fit = map(trend, broom::glance)) %>%
  tidyr::unnest(model_fit)
```

```
## # A tibble: 142 x 15
##    country continent data  trend r.squared adj.r.squared sigma statistic
##    <fct>   <fct>     <lis> <lis>     <dbl>         <dbl> <dbl>     <dbl>
##  1 Afghan~ Asia      <tib~ <lm>    0.00539       -0.0941  113.    0.0542
##  2 Albania Europe    <tib~ <lm>    0.678          0.646   710.   21.1
##  3 Algeria Africa    <tib~ <lm>    0.782          0.760   641.   35.9
##  4 Angola  Africa    <tib~ <lm>    0.129          0.0422 1141.    1.48
##  5 Argent~ Americas  <tib~ <lm>    0.706          0.677  1059.   24.0
##  6 Austra~ Oceania   <tib~ <lm>    0.969          0.966  1432.  318.
##  7 Austria Europe    <tib~ <lm>    0.996          0.996   620. 2654.
##  8 Bahrain Asia      <tib~ <lm>    0.866          0.852  2081.   64.5
```

```
##  9 Bangla~ Asia      <tib~ <lm>    0.649          0.614   146.   18.5
## 10 Belgium Europe    <tib~ <lm>    0.993          0.992   728. 1453.
## # ... with 132 more rows, and 7 more variables: p.value <dbl>, df <int>,
## #   logLik <dbl>, AIC <dbl>, BIC <dbl>, deviance <dbl>, df.residual <int>
```